A Method for Optimum Control of Dynamic Load Distribution in Time-sensitive Communication Networks for Manufacturing Automation

Thomas Weichlein

A thesis submitted to
The University of Gloucestershire
in accordance with the requirements of the degree of
Doctor of Philosophy
in the School of Computing and Engineering

June 2024

Word Count: 73707

Abstract

Communication between end stations in a contemporary communication network typically occurs over several paths for media redundancy and throughput enhancement. To use all paths as evenly as possible, load balancing or load distribution methods are applied, mainly in higher level information technology (IT) networks such as Internet Service Provider (ISP) networks, campus networks, and mobile access networks. Automation networks of operations technology (OT), particularly Manufacturing Automation Networks (MAN), have rarely been subject to load distribution control. IEEE Time Sensitive Networks (TSN) are a relatively young network development that offers, among other features, redundant paths for automation networks, the essential precondition for load distribution. Furthermore, the TSN framework defines several traffic shapers and schedulers which are expected to have different impacts on Load Distribution Control (LDC). However, the selection of the right traffic shaper or scheduler for an automation network is challenging. Their influence depends on various network parameters such as network extension, network cycles, application cycles, and the amount of data per traffic class and per network cycle. **This thesis proposes, designs, and develops a method for optimum control of dynamic load distribution in time-sensitive communication networks for manufacturing automation.**

The research philosophy underlying this research project is positivism. Literature review (textual analysis) is used to obtain secondary data on relevant use cases of automation communication, control theory concepts, and network standards. To obtain the primary data on the control results, simulation is used.

Firstly, the influence of different TSN MAN network parameters and properties on LDC is investigated. Secondly, the data flow control is analysed as a subsequent control task for LDC under the influence of the different traffic shapers and traffic schedulers. Based on these results, thirdly, a dedicated optimum load distribution control method for MAN with a single automation controller (AC) is proposed. Then, this optimised LDC method is applied to a network with multiple ACs.

Contribution: The results of the analysis and evaluation of the influence of the various automation parameters as well as the TSN shapers and schedulers provide a detailed picture of the data flow control options within TSN MAN. The derivation of control properties for data flow control and load distribution control as well as their control simulation and evaluation of the results create the prerequisites for the design of LDC solutions in these networks. A strong influence of the application cycles on the control dynamics and stability is demonstrated. It is further shown that network nodes using SPQ, SPQ with Preemption, and EST provide rather low path delays. They are the best shaper and scheduler selections in high dynamic networks and in larger networks. The application of network nodes using CQF and ATS can result in significantly high path delay times, and thus, high control dead times, especially in larger networks with a high hop count. Therefore, they are recommended only for smaller network sizes with lower dynamic requirements. Based on this preliminary work, the study also provides optimised control solutions for single and multiple AC LDCs, both with and without mutual AC interference. The results based on network simulations confirm the suitability of these solutions and show that the overall convergence time improves.

Thus, the present study provides a new comprehensive view and solutions to the possibilities of LDC within TSN MAN that have been lacking in the research literature so far.

Declaration of Original

I declare that the work in this thesis was carried out in accordance with the regulations of the University of Gloucestershire and is original except, where indicated by specific references in the text. No part of the thesis has been submitted as part of any other academic award. The thesis has not been presented to any other education institution in the United Kingdom or overseas. Any views expressed in the thesis are those of the author and in no way represent those of the University.

Signed: 

Date: 2024/06/04

DOI: 10.46289/77UV2F4D

Acknowledgment

At this point I would like to express my special thanks to my first supervisor, Prof Shujun Zhang from the University of Gloucestershire for his advice and guidance, which helped me to get this thesis on the right track. His experience, scientific knowledge and domain expertise were invaluable for its success. I would also like to thank my second supervisor, Dr Xu Zhang from the University of Southampton, for his support and critical review of my work.

I also thank Prof Ros Jennings, Dr Philippa Ward, and Dr John Hockey from the University of Gloucestershire for the transfer of the necessary basics of research philosophy, methods, and methodology. Many thanks also to Dr William Sayers for the critical review of the research design and to the staff of the university for their support during these years.

I am also very grateful for the many inspiring and informative discussions with my system architecture colleagues at SIEMENS AG, Digital Industries, during the many years with definitions of general automation communication requirements and during the preparation of contributions to IEC and IEEE communication networks standardisation.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AC	Automation Controller
ACO	Ant Colony Optimisation
ACOC	Ant Colony Optimisation Control
ACID	Automation Controller Identifier
AD	Acyclic Data
AI	Artificial Intelligence
AIMD	additive-increase/multiplicative-decrease
APPID	Application Identifier
AR	Application Relationship
ARP	Address Resolution Protocol
AS	Autonomous System
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
ATS	Asynchronous Traffic Shaper
ATS-DLDC	Distributed Load Distribution Control based on ATS
AV	Audio Video
AVB	Audio Video Bridging
Avnu	Automation Alliance for TSN and AVB application promotion
BE	Best Effort (data traffic)
BLCE	Bridge Local Computation Engine
CBS	Credit Based Shaper
CCLB	Cloud Computing Load Balancing
ccw	Counterclockwise
CD	Control Data
CIR	Configuration in Run
CLDC	Central Load Distribution Control
CMA	Cumulative Moving Average
CNC	Central Network Controller
COMET	Programming Language designed for Optimisation Problems

CPU	Central Processing Unit
CQF	Cyclic Queuing and Forwarding
CQF-DLDC	Distributed Load Distribution Control based on CQF
cw	Clockwise
DC	Distribution Control
DCB	Data Centre Bridging
DL	Deep Learning
DLB	Dynamic Load Balancing
DLDC	Distributed Load Distribution Control
DSLB	Distributed Systems Load Balancing
ECT	Explicit Equal Cost Tree
EIGRP	Enhanced Interior Gateway Routing Protocol
ES	End Station
EST	Enhancements for Scheduled Traffic
EST-DLDC	Distributed Load Distribution Control based on EST
ETS	Enhanced Transmission Selection
FC	Flow Control
FDB	Filtering Data Base
FID	Forwarding Database Identifier
FP	Frame Preemption
FRER	Frame Replication and Elimination for Reliability
FTP	File Transfer Protocol
gPTP	Generic Precision Time Protocol
GMPLS	Generalized MPLS
GNU	GNU is not Unix
I-CD	Isochronous Control Data
HMI	Human Machine Interface
HSR	High Availability Seamless Redundancy
ICI	In-Class-Interference
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers

IET	Interspersing Express Traffic
IGP	Interior Gateway Protocol
I/O	Input/Output
IP	Internet Protocol
IS-IS	Intermediate System – Intermediate System
ISO	International Organization of Standardization
IPV	Internal Priority Value
ISP	Internet Service Provider
IT	Information Technology
LA	Listener Attach
LAN	Local Area Network
LDC	Load Distribution Control
LNI	Labs Network Industrie 4.0
LSP	Labeled Switched Path
LTE	Long Term Evolution mobile networks
LTE-A	LTE Advanced network or 4G+ network
LR	Literature Review but also Listener Ready, context dependent
MAC	Media Access Control
MAN	Manufacturing Automation Networks
MC	Motion Controller
MIB	Management Information Base
ML	Machine Learning
MLC	Machine Learning Control
MLTE	Multilayer Traffic Engineering
mmWave NR	Millimeter wave new radio (30 to 300 GHz)
MPLS	Multi-Protocol Label Switching
MRP	Media Redundancy Protocol but also Multiple Reservation Protocol
MRT	Maximally Redundant Tree
MSRP	Multiple Stream Registration Protocol
MSTP	Multiple Spanning Tree Protocol
MSTI	Multiple Spanning Tree Instance

M2M	Machine to Machine
NARMA	Non-linear Auto-Regressive Moving Average
NETCONF	Network Configuration Protocol
NI-CD	Non-isochronous Control Data
NLB	Network Load Balancing
NLC	Network Load Control
NP-hard	Non-deterministic Polynomial-time hardness
ns-3	Network Simulator Tool
OMNet++	Network Simulator Tool
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OPC	Open Platform Communications
OPC UA	Open Platform Communications Unified Architecture
OPC UA PubSub	OPC UA Publisher Subscriber
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OT	Operations Technology
PBB	Provider Backbone Bridging
PBBN	Provider Backbone Bridged Networks
PC	Packet Control
PCA	Path Computation Agent
PCE	Path Computation Element
PCR	Path Control and Reservation
PHY	Physical access to the communication media
PLC	Programmable Logic Controller
PN	PROFINET
PRP	Parallel Redundancy Protocol
PSFP	Per-Stream Filtering and Policing
PTP	Precision Time Protocol
PT1	Proportional Element with one time constant
PID	Proportional Integral Differential (Controller)

QoS	Quality of Service
RAP	Resource Allocation Protocol
RIP	Routing Information Protocol
RR	Resource Reservation
RSTP	Rapid Spanning Tree Protocol
RTT	Round Trip Time
RFC	Request for Comments
RM	Rolling Mean
RO	Research Objective
SDN	Software Defined Networks
SDU	Service Data Unit
SLB	Server Load Balancing
SNMP	Simple Network Management Protocol
SPB	Shortest Path Bridging
SPBM	Shortest Path Bridging – MAC
SPBV	Shortest Path Bridging – VID
SPQ	Strict Priority Queuing or Static Priority Queuing
SPQ-DLDC	Distributed Load Distribution Control based on SPQ
SR	Stream Reservation
SRP	Stream Reservation Protocol
ST	Scheduled Transmission
TA	Talker Advertise or Talker Announce
TAS	Time Aware Shaper, synonym for EST
TCAM	Ternary Content Aware Memory
TCP	Transport Control Protocol
TE	Traffic Engineering
TED	Traffic Engineering Database
TLV	Type Length Value coding
TSN	Time sensitive networks
UDP	User Datagram Protocol
UML	Unified Modelling Language

URL	Unified Resource Locater
VID	VLAN Identifier
VLAN	Virtual Local Area Network
VxLAN	Virtual Extensible LAN
WDM	Wavelength Division Multiplexing
WAN	Wide Area Network

Definitions

Distribution Controller	*The controller which determines the target load distribution, and which feeds the flow controller for the individual paths. It is part of the Distribution Control Assembly.*
Distribution Control Assembly	*The assembly consisting of the actual distribution controller and the downstream flow controller.*
Flow Controller	*The controller which controls the increase and decrease of load on the individual network paths and which is fed by the distribution controller. It is part of the Distribution Control Assembly.*
Load balancing	*Distribute communication load equally among several communication paths.*
Load distribution	*Distribute communication load onto several communication paths according to a given distribution requirement. A more general expression for load sharing and load balancing.*
Control plane	*The functional entities of bridges and bridged end stations providing interfaces for network protocols to influence the behaviour of these devices.*
Data plane	*The functional entities of bridge or bridged end stations that provide mechanisms for the data transport.*
Gating window	*Time duration when a queue gate at a port is open. Synonym for gate open window.*
Load sharing	*Synonym for load balancing.*
Intra-domain	*Within and bound to a delimited network domain.*
Network cycle	*Synonym for network gating cycle or gating cycle.*

Switch	*In the network domain terminology often used as a synonym for bridge.*

According to IEEE 802.1Q:

End station	*A device attached to a LAN or MAN, which acts as a source of, and/or destination for, data traffic carried on the LAN or MAN.*
Bridge	*A functional unit that interconnects two or more LANs or MANs that use the same Data Link layer protocols above the MAC sublayer but can use different MAC protocols. Synonym for switch.*
Gating cycle	*The period of time over which the sequence of operations in a gate control list repeat. It is here and in other literature also addressed as network gating cycle or shortly network cycle.*
Listener	*The end station that is the destination, receiver, or consumer of a stream.*
Stream	*A unidirectional flow of data (e.g., audio and/or video) from a Talker to one or more Listeners.*
Talker	*The end station that is the source or producer of a stream.*
Time Sensitive Stream	*A stream of data frames that are required to be delivered with a bounded latency.*
Traffic Class	*A classification used to expedite transmission of frames generated by critical or time sensitive services. Traffic classes are numbered from zero through N-1, where N is the number of outbound queues associated with a given Bridge Port, and 1*

	<= N <= 8, and each traffic class has a one-to-one correspondence with a specific outbound queue for that Port. Traffic class 0 corresponds to nonexpedited traffic; nonzero traffic classes correspond to expedited classes of traffic. A fixed mapping determines, for a given priority associated with a frame and a given number of traffic classes, what traffic class will be assigned to the frame.
Stream Reservation (SR) Class	*A traffic class whose bandwidth can be reserved for e.g., audio/video (AV) traffic. A priority value is associated with each SR class. SR classes are denoted by consecutive letters of the alphabet, starting with A and continuing for up to seven classes.*
StreamID	*A 64-bit field that uniquely identifies a stream.* *StreamID comprises the following two subcomponents:* *A 48-bit MAC Address associated with the System sourcing the stream to the bridged network. The MAC address shall be unique within the network.* *A 16-bit unsigned integer value, **Unique ID**, used to distinguish among multiple streams sourced by the same System.*

Chapter 1 Introduction

1.1 Overview

This chapter provides background on how the research project is embedded in the current technology of industrial communication networks and how the discipline of control engineering contributes to possible solutions for load distribution. It also explains why this project is important, its overall aim and research objectives, the contributions to the new knowledge generation, and how the thesis is structured.

1.2 Project Background

The continuously increasing communication demand in the industry is primarily due to the "Industry 4.0" industrial revolution. This implies a significant expansion in the digitalisation of the production process and vertical communication connectivity from cloud-based servers down to the sensor level in an industrial plant. This increase implies not only a growing demand for data volume and communication speed, but also a higher need for reliable and deterministic data transport. These developments have led in a first step to the development of the Audio Video Bridging (AVB) standard (IEEE 802.1BA, 2011) and finally to the creation of a "Time-Sensitive Networks (TSN)" (Finn, 2018; Lo Bello & Steiner, 2019) Task Group (TG) as part of the IEEE 802.1 Working Group (WG). TSN is defined by the associated IEEE standards, some of which are still being developed and extend the IEEE 802.1 standards:

- Forwarding and Queuing Enhancements for Time-Sensitive Streams (IEEE 802.1Qav, 2009),
- Enhancements for Scheduled Traffic (EST) (IEEE 802.1Qbv, 2015),
- Frame Preemption (FP) (IEEE 802.1Qbu, 2015),
- Path Control and Reservation (PCR) (IEEE 802.1Qca, 2015),
- Per-Stream Filtering and Policing (PSFP) (IEEE 802.1Qci, 2016),
- Cyclic Queuing and Forwarding (CQF) (IEEE 802.1Qch, 2019),

- Frame Replication and Elimination for Reliability (FRER) (IEEE 802.1CB, 2017),
- Stream Reservation Protocol (SRP) Enhancements and Performance Improvements (IEEE 802.1Qcc, 2018),
- Link-local Registration Protocol (LRP) (IEEE 802.1CS, 2019)
- Timing and Synchronization for Time-Sensitive Applications (gPTP) (IEEE 802.1AS, 2020),
- Asynchronous Traffic Shaper (ATS) (IEEE 802.1Qcr, 2020),
- Resource Allocation Protocol (RAP) (IEEE 802.1Qdd, 2023).

Thereby, TSN defines various new functionalities and different traffic shapers and schedulers, such as the Credit Based Shaper (CBS) (IEEE 802.1Qav, 2009; IEEE 802.1Qcc, 2018), EST, CQF, and ATS. It also allows the extension of the classical Strict Priority Queuing (SPQ) (IEEE 802.1Q, 2022) with stream reservation (SR) or bandwidth resource reservation (RR) (IEEE 802.1Q, 2022; IEEE 802.1Qdd, 2023), and frame preemption (FP) (IEEE 802.1Qbu, 2015; IEEE 802.3br, 2016), making SPQ more deterministic and faster. The goal of a TSN is to achieve highly efficient and deterministic data transport (Nasrallah et al., 2019). TSN also allow for the use of multiple communication paths, primarily to provide seamless media redundancy according to IEEE 802.1CB (IEEE 802.1CB, 2017), which defines "Frame Replication and Elimination for Reliability (FRER)."

Similar to TSN networks for industry, increasingly loaded network paths have also occurred earlier in general IT networks, such as Internet or campus communication networks. This often led to network congestion and hence data loss. When these networks were initially not set up redundantly, that is, when only one communication path was available, researchers applied congestion control solutions to cope with the increasing congestion loss of data traffic. The research community differentiates between open-loop and closed-loop congestion control.

With open-loop congestion control (Wu & Mark, 1993), the approach is to stretch ingress traffic. This can be achieved, for example, according to a principle called “Leaky Bucket principle”. This symbolises that traffic bursts are converted into a constant data rate.

A different approach to congestion control is closed-loop congestion control. Here, congestion control becomes active only when congestion on a network path actually occurs. Early closed-loop congestion control was applied in combination with the widely used Open Systems Interconnection (OSI) layer 4 transport control protocol TCP (IETF RFC 793, 1981) which led to a variety of TCP congestion control algorithms (Hasegawa et al., 2000; IETF RFC 5681, 2009).

Since the late 1990s, general IT networks for Internet or campus communications, both wired and wireless, have been increasingly set up as multi-paths networks. In addition to the advantages of redundancy, the availability of multiple paths allows the use of load sharing, load balancing, or load distribution concepts. All three expressions are used as equivalent terms in the research literature and in this thesis.

The difference between load balancing and congestion control is that load balancing can shift load peaks to an alternate path. In the ideal case, the result is that all available paths are evenly loaded. The major advantage is that traffic does not need to be delayed to resolve upcoming congestion. This makes load balancing especially interesting in combination with time critical control data (CD) for automation purposes.

Load balancing is not only found in different network areas such as ISP networks (Wang et al., 2006), campus networks (Elwalid et al., 2002), or access networks for mobile connectivity (Ahmad et al., 2015), but is also used in combination within different IT areas. The literature provides solutions in the area of Server Load Balancing (SLB) (Bojović & Živko, 2022; Cardellini et al., 1999; Wilson & Deepalakshmi, 2019), Distributed Systems Load Balancing (DSLB) (Grosu & Chronopoulos, 2005; Metawei et al., 2012; Taley & Keole, 2015; Zaki et al., 1996), Cloud Computing Load Balancing (CCLB) (Katyal & Mishra, 2014; Shahid et al., 2020; Tawfeeg et al., 2022; Zhang & Zhang, 2010) and Network Load Balancing (NLB) (Ahmad et al., 2015; Antic et al., 2010; Chadha & Gupta, 2013; Elwalid et al., 2002; Fortz & Thorup, 2000) .

SLB, DSLB, and CCLB all three have in common that their primary goal is to distribute the load on the systems rather than on the network, which is the goal

of the NLB. However, they also imply NLB as a side effect, depending on the network topology of the systems. Controlling the network load is also the focus of this study. Thus, the approaches applied within these load balancing techniques can yield valuable insights. Furthermore, they are interesting because their controller types can, in principle, also be used for NLB.

NLB is typically used in the layer 3 routing technology (Tanenbaum et al., 2021). Their dedicated load balancing methods include those for ISP networks (Wang et al., 2006), campus networks (Elwalid et al., 2002), and access networks for mobile connectivity (Ahmad et al., 2015).

However, it is also applied in layer 2 networks such as Data Center Bridging (DCB) (Perry et al., 2014; Shuo et al., 2016; Wei et al., 2014; Zhang et al., 2018) or Software Defined Networks (SDN) (Jahde et al., 2021; Todorov et al., 2020).

The applied controller type is a central and crucial element of all load balancing solutions. This must be adapted to network characteristics. All types of controllers have been used in previous research, such as linear control (Kandula et al., 2007; Wang et al., 2006), stochastic control (Neely et al., 2008), fuzzy control (Pompili & Priscoli, 2008; Talaat et al., 2019; Wang & Hung, 2012), Smith predictive control or model predictive control (Mascolo, 2000; Quang et al., 2020), ant colony control (Mohammadnia et al., 2016; Zhang & Zhang, 2010), neural network control (Talaat et al., 2019; Wang & Hung, 2012), dedicated algorithm control (Elwalid et al., 2002; Farahmand et al., 2005), and control by Artificial Intelligence (AI) or Machine Learning (ML) (Anna Victoria Oikawa et al., 2020; Todorov et al., 2020).
The selection and application of the right controller type for network load distribution control in TSN MAN is one of the tasks of this research.

Another dimension of load balancing is that it comprises three control tasks. These are (Ahmad & Khan, 2018; Elwalid et al., 2002; Lopez-Perez et al., 2016; Neely et al., 2008; Wang et al., 2006):

- Flow control: The control algorithm to control the data flow on a single path to increase or decrease throughput.

- Fairness control: The control algorithm to regulate the fair distribution of the reduction or increase in throughput among the different data flows.
- Distribution control: The control algorithm to allocate parts of a stream or several streams evenly to a choice of paths.

Fairness control is not relevant for automation network CD, as CD data flows are typically only allowed to be minimally delayed. However, flow control and distribution control are important for load distribution control in a TSN MAN.

In factory automation applications, networks with smaller spatial extensions are used to transport information between automation controllers (AC) and devices, such as drives, sensors, and actuators. They are typically based on the OSI layer 2 technology using switching (IEC/IEEE 60802, 2018; Tanenbaum et al., 2021).

To achieve redundant connections with minimum wiring effort, ring topology has become a prevalent topology in redundant industrial automation networks. Figure 1.1 shows a typical industrial automation network setup in which several field-level rings are redundantly coupled to a controller-level ring. This, in turn, is redundantly coupled to a higher-level Information Technology (IT) or Operational Technology (OT) network (IEC/IEEE 60802, 2018).

Controller-level rings usually contain a variety of higher-level ACs such as programmable logic controllers (PLC) or motion controllers (MC). However, a field-level ring typically consists of only one AC that controls a variety of automation devices, such as drives, sensors, actors, or decentral peripherals providing digital and analog inputs and outputs. Field-level ACs communicate with controller-level ACs.

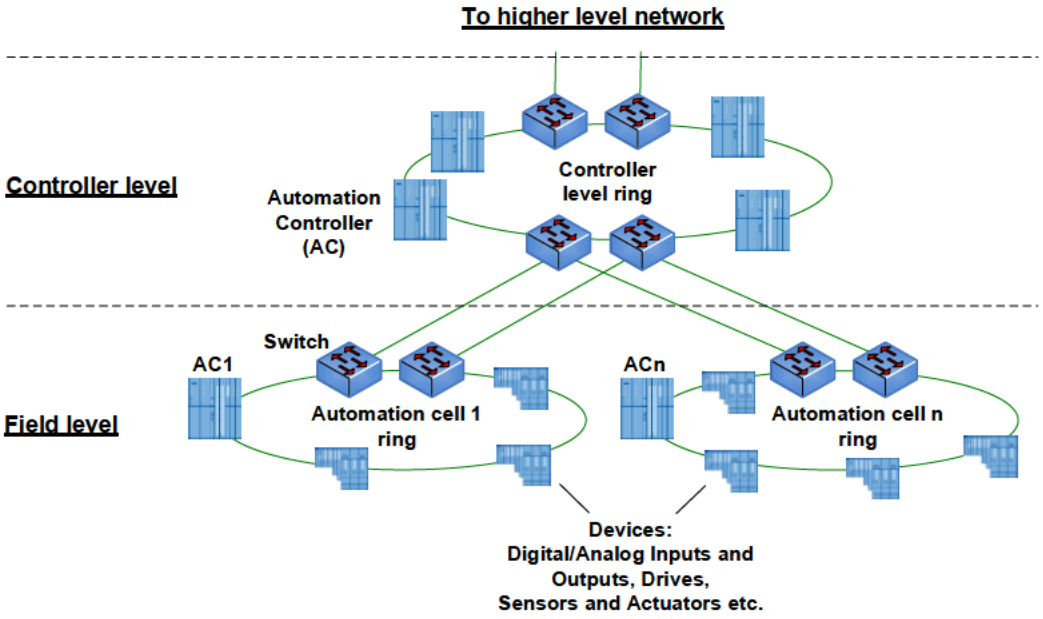


Figure 1.1: Typical automation communication network setup

To date, communication connections in automation networks have typically been set up redundantly, primarily because of fail-safety rather than load sharing.

Comparing legacy layer 3 networks, three major load distribution strategies, sometimes known as traffic engineering concepts, are visible (Wang et al., 2006):

1. oblivious routing, that is, routing on a fixed scheme without incorporating actual changes in the network load along various data paths.
2. traffic control using predicted traffic demands based on recorded traffic history.
3. adaptive or dynamic control using metrics on traffic load along the available data paths.

The first two methods, that is, oblivious and predictive traffic control, are advantageous particularly in conjunction with uncertain network demand estimations. However, the communication demands of automation applications are well-defined and predictable within certain limits in the network setup. Because of this advantage, MAN can be subject to both traffic distribution planning during the network setup phase and adaptive or dynamic traffic control at runtime.

Adaptive or dynamic control is based on routing decisions using network-load metrics. A control algorithm constantly controls the load distribution on several paths to achieve an optimal or near-optimal load distribution and to minimise the local load maxima. Subsequent flow controllers have the task of increasing or decreasing the load on single paths following the distribution calculation from the distribution controller (Ahmad et al., 2015; Lopez-Perez et al., 2016; Neely et al., 2008; Wang et al., 2006)

Regarding distribution and flow control, more research has been conducted on distribution control (Ahmad et al., 2015; Lopez-Perez et al., 2016; Neely et al., 2008; Wang et al., 2006) than on the flow control subtask (Bonomi & Fendick, 2002; Jain, 1998), which is identical with closed-loop congestion control for single path networks. However, certain network characteristics play a crucial role in flow control.

One important influencing factor is the cycle time of the automation application tasks hosted by the ACs. These application cycle times, sending data at each cycle, define the minimum rolling mean interval of a load measurement. This, in turn, forms a system time constant and thus limits attainable control performance, that is (Goodwin et al., 2001; Normey-Rico & Camacho, 2007), the time to establish a new load distribution setpoint.

The other important parameter is the underlying basic cycle time of the network communication, which must be long enough to transport the maximum amount of data but small enough to serve the fastest application in the network domain.

Furthermore, the network extension and applied traffic shaper or scheduler influence the path delays. These represent dead time elements that characterise the flow control circuit properties.

The influence of these parameters: application cycle, communication cycle, shaper influence, and network extension, is typical for TSN automation networks. Their impact is particularly high on sophisticated MAN TSN owing to their fast automation applications such as motion control or packaging and labeling operations. These problems are not a big issue in general IT networks though and

research has thus not yet investigated their effect on the design of data traffic distribution control and flow control. In addition, the influence of RR and FP on these tasks has not been investigated either.

Therefore, in the first step, the influence of different network parameters, namely control location design possibilities, relevant network topologies, control setup design possibilities and characteristics, eligible traffic classes, automation applications influence, the role of stream reservation, and error mitigation strategies, is analysed.

In the second step, data flow control for control data within TSN automation networks is investigated under the influence of these parameters on control dynamics and stability. Furthermore, the impact of bandwidth reservation and frame preemption is analysed and recommendations for load measurements are provided. Thus, the study provides the network designer with a valuable tool to select the appropriate TSN mechanisms with the aim of deploying LDC adapted to the automation applications.

A new method for distribution control in a TSN MAN is proposed. It specifically addresses the problem of different application cycle times for various applications in the network domain. Dedicated distribution controllers for a range of application cycles are proposed. These improve the control dynamics and reduce the effort required to reconfigure the control configuration in response to network setup changes.

Then, this new load distribution method is described and investigated in terms of its dynamic capabilities and behaviour in multi-talker/multi-listener applications. The optimisation potential is analysed and evaluated. It shows that the dependency of the load distribution convergence on the longest application cycle can be resolved, and the overall load distribution convergence improves.

Furthermore, solutions addressing the use of multiple ACs in a ring are provided. A solution using a control sovereignty passing method for multiple ACs with mutual dependency is proposed. This is required when SPQ is selected as a shaper basis. The second solution addresses multiple ACs without mutual dependency

which is the case with EST, CQF, or ATS. This is characterized by the temporal decoupling of the control processes.

These new control methods can be applied in manufacturing automation network controller level rings and field level rings. The new control methods improve network utilisation and reduce the probability of network overload situations.

Thus, the present study offers a new and first comprehensive design guide for the full implementation of LDC in TSN MAN in terms of:

1. The selection of the TSN shaper depending on the automation applications and network parameters.
2. The design of the data flow and load distribution controller.
3. Choosing solutions to cope with multiple ACs in the TSN LDC domain for the different types of TSN.

1.3 Motivation

Until now, there has been limited theoretical research and practical application work on efficient load sharing and load balancing over multiple paths of TSN, which is particularly true in industry. Nayak (2018) investigated general network traffic distribution possibilities applying a central approach in combination with EST. It however achieves only a limited dynamic performance because of necessary high-effort distribution re-calculations. Ojewale and Yomsi (2020) presented two other dedicated centrally computed routing algorithms to optimise a combination of path lengths and loads. Here, too, the computational effort and time increase disproportionately with the number of network nodes due to the central approach. MAN, however, have high dynamic requirements regarding reaction to load changes. Therefore, these, and particularly the more recent TSN MAN, offer new grounds for research on network load distribution, which can be expected to contribute to enhancing the performance of these networks. This includes not only distribution control but also subsequent flow control per path.

Especially in larger networks, the increasing number of end stations in an automation ring also increases the amount of CD to be transported over the two possible paths between communication participants. However, the presence of fast applications demands short communication cycles which is particularly true for EST and CQF traffic shapers which have only a limited CD transport capacity in their communication cycle. This conflict forces the network designer to optimally use both the available paths in an automation ring. Thus, the use of dynamic load distribution will be an important means of optimum use of the network preconditions.

A further important reason for load distribution in TSN MAN is the limited possibility for seamless communication, that is, double sending over two paths for redundancy reasons. The reason is, that there are only limited expensive hardware resources for duplicate filtering in the switch ASICs. This increases the amount of single-path CD, and thus, the need for load distribution.

The main purpose and motivation of this study is therefore to evaluate a possible extension of the current content of the TSN project by proposing a method for distribution control and flow control. This will optimise the use of multiple TSN communication paths to dynamically distribute data traffic on the data paths within manufacturing automation communication networks.

1.4 Overall Aim and Research Objectives

The overall aim is to propose, design, develop, and validate a method for optimum control of dynamic load distribution in time-sensitive communication networks for manufacturing automation**.**

The proposed method is verified and validated by control circuit simulations and network simulations.

The following research objectives are addressed:

1. To analyse the properties of TSN networks that influence the goal of establishing a load distribution.

2. To analyse and evaluate the influence of different TSN traffic shapers and schedulers on subsequent data flow control.
3. To propose, design, develop, and validate an optimised closed loop load distribution control method that can be effectively and efficiently applied in the currently evolving Layer 2 TSN for different types of manufacturing automation networks.
4. To extend the optimised closed loop load distribution control method to support multiple automation controller setups.

1.5 Thesis Contributions to New Knowledge Generation

This thesis' main contribution to the new knowledge generation is to demonstrate the possibilities of load distribution in redundant TSN manufacturing automation networks. The following contributions are made during the course of this task:

1. An analysis of TSN automation networks with regard to:
    a) control location design possibilities,
    b) relevant network topologies,
    c) eligible traffic classes,
    d) control aspects,
    e) the influence of automation applications,
    f) the influence of stream reservation,
    g) and network error mitigation strategies.

    Thereby, a detailed picture of load distribution control possibilities within TSN MAN is provided.
2. Derivation of a closed-loop load distribution control model for automation ring networks. The influence of the different types of TSN traffic shapers and scheduler on data flow control is demonstrated. The different types of TSN network communication paths are simulated and evaluated in terms of controllability and stability.
3. Recommendations for the selection of traffic shapers or traffic schedulers considering the types of automation tasks.
4. Proposal of an optimised control method for load distribution in TSN MAN.

5. The proposal of an optimised control strategy for TSN MAN containing multiple automation controllers.

Part of the bullet points 1 to 3 has been published (Weichlein et al., 2023).

1.6 Publications

Partial results of this thesis have been published (Weichlein et al., 2023):

> Weichlein, T., Zhang, S., Li, P., & Zhang, X. (2023). Data Flow Control for Network Load Balancing in IEEE Time-Sensitive Networks for Automation. *IEEE Access*.

1.7 Thesis Structure

This thesis is structured as follows.

After this introduction, the literature review chapter provides an overview of communication networks, in general, and for automation. It also covers the basics of the control theory for network load control and its application in network congestion control, traffic engineering and network load balancing.

Chapter 3, “Research Methodology and Design”, provides the details of the research process and the methods for data collection and presentation.

Chapter 4, “The Influences of TSN MAN Properties on Load Distribution Control”, provides a detailed investigation into load control relevant properties of TSN automation networks. Here, relevant network topologies, load distribution design possibilities, eligible traffic classes, plant properties and control aspects are discussed. Furthermore, the influences of the automation applications and of stream reservations are analysed. The consequences of network errors are discussed.

Chapter 5, “Application of Different TSN Traffic Shapers and Schedulers for Subsequent Data Flow Control”, examines the influence of the different TSN traffic shapers and traffic schedulers on the system properties, that is, the network path as the system under control. Furthermore, in this chapter, these influences are simulated and discussed with regard to their influence on the controllability and stability of the control.

Chapter 6, “A New Control Method for Load Distribution Optimisation in TSN MAN,” builds on the findings of Chapters 4 and 5 and firstly discusses and selects an appropriate core controller type for flow control and distribution control. In the second step the drawbacks of the classical approach of controlling the data flow and introducing a better control strategy for an optimised control are addressed. Furthermore, suitable feedback generation methods are discussed and proposed. Finally, at the end of this section, the optimised method is verified by network simulations.

Chapter 7, “Extension of the LDC Optimisation to Support Multiple Automation Controllers,” extends the new optimised control method from Chapter 6 to apply to multiple independent automation controllers with and without mutually influencing each other.

Chapter 8, “Conclusion and Further Work,” concludes the thesis, summarises the research results, and discusses the original contribution. Furthermore, it discusses the limitations and the further work for improving this research on load distribution in manufacturing automation networks.

Chapter 2 Literature Review

2.1 Introduction

This chapter provides a critical literature review with a focus of load control in the areas of TSN, their application as MAN, control theory for load control in communication networks, and the related main control methods. It provides an overview of the current knowledge in these areas. It also aims to identify the gap in the existing research on network traffic load control in MANs based on TSN.

2.2 Communication Networks

This section provides an overview of load reduction research in general Ethernet communication networks (IEEE 802.1Q, 2022; Tanenbaum et al., 2021), the special properties of MAN (IEC/IEEE 60802, 2018; IETF RFC 8578, 2019), and the application of TSN (IEEE 802.1Q TSN TG, 2022) in MAN. It thus clarifies the technical basis on which the research is based.

2.2.1 General Communication Networks

General communication networks build the basis for TSN MAN and consist of data relaying elements called routers and switches, which are connected via wire-based or wire-less connections for data transport. It is generally accepted to distinguish the data processing systems of manufacturing processes as operation technology (OT) from pure information technology (IT).

The state-of-the-art model to describe communication functions in technical or computer systems is the Open Systems Interconnection (OSI) reference model (ISO/IEC 7498-1, 2000), which was defined by the International Organization of Standardization (ISO) and is depicted in Figure 2.1.

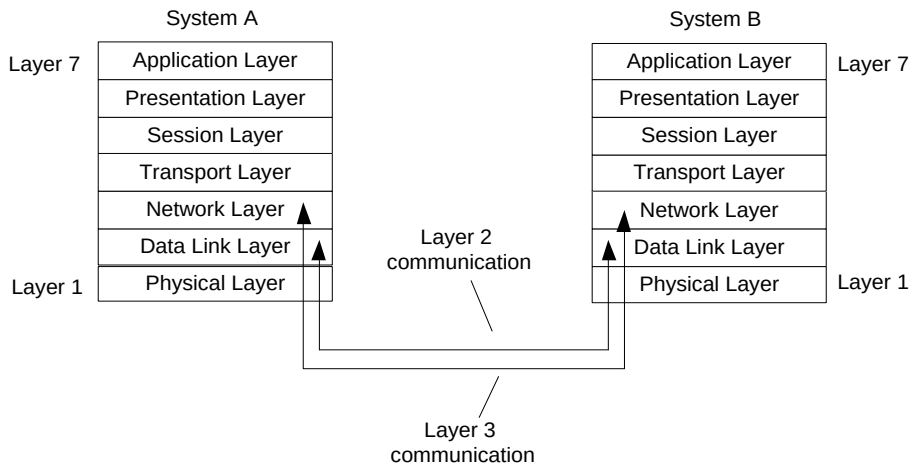


Figure 2.1: Open Systems Interconnection (OSI) communication model

The ISO/OSI reference model divides a communication system into seven hierarchical layers (Tanenbaum et al., 2021):

- Layer 1: the physical layer (PHY)
- Layer 2: the data link layer containing the media access control (MAC)
- Layer 3: the network layer
- Layer 4: the transport layer
- Layer 5: the session layer
- Layer 6: the presentation layer
- Layer 7: the application layer

For this research, the layer 2 data link layer and the layer 3 network layer research results on traffic load reduction and distribution are important. These are of interest for the mechanisms of data packet transport from one network participant to another. This is independent of whether physical Layer 1 of the network is based on wire, as with campus or Internet networks, or is wireless as in the case of mobile networks.

The communication links between the end stations and relaying elements have limited data rate capacities which can lead to network congestion. This problem was early addressed by Chiu and Jain (1989) in general communication layer 3 networks, mainly in combination with the layer 4 TCP as the main transport protocol. At that time, increasing network loads stuffed networks and delayed the

response times of client requests to network services. They investigated throughput increase/decrease algorithms and demonstrated that they are suitable for congestion reduction.

To avoid network congestion or achieve a homogenously distributed load on individual network paths, either a means of congestion control or, in combination with multiple network paths, a load distribution control mechanism is necessary. Extensive research has been conducted on load sharing, load balancing, and load distribution solutions. All three concepts are mostly used synonymously. These operate on OSI Layer 3, such as on ISP networks or campus networks, or within small Layer 2 sub-networks, or only between pairs of single nodes such as clients and servers.

The literature review shows that existing load controlling concepts within various communication networks technologies can generally be classified as follows:

1. Network congestion control (Kanagarathinam et al., 2020; Kasoro et al., 2021; Katabi et al., 2002; Wu et al., 2009), aims to reduce the transmission of traffic on the sender side to avoid congestion. It must be applied if no alternate path for load distribution is available.
2. Traffic engineering (Elwalid et al., 2002; Fortz et al., 2002; Lemeshko et al., 2013; Santos et al., 2009; Zhang et al., 2018) has the goal to plan all traffic properly from the beginning. It can also include to dynamically react to changes in the network.
3. Load balancing, which again can be subdivided into:
    a. Server load balancing (Bojović & Živko, 2022; Cardellini et al., 1999; Wilson & Deepalakshmi, 2019),
    b. Distributed systems load balancing (Grosu & Chronopoulos, 2005; Metawei et al., 2012; Taley & Keole, 2015; Zaki et al., 1996),
    c. Network load balancing, both open and closed-loop, which is often used as a synonym for network load control (see bullet point 4 below for references),

d. Cloud computing load balancing (Ahmad & Khan, 2018; Katyal & Mishra, 2014; Nezami et al., 2021; Rajeshkannan & Aramudhan, 2016; Shahid et al., 2020; Zhang & Zhang, 2010);

4. Closed-loop network load distribution control or balancing (Han et al., 2021; Jahde et al., 2021; Kandula et al., 2007; Lemeshko et al., 2013; Mohammadnia et al., 2016).

Having OT MAN networks in focus, which typically do not feature several servers or a widely distributed system, especially the closed-loop network load distribution control concept, is the most important for this research project. However, the results of the referenced research on other load distribution concepts within the areas of network congestion control, traffic engineering and load balancing are also relevant. They provide valuable data and experience on sensible control concepts in various network areas. This data forms the basis for this research on dynamic load distribution control in TSN MAN, a topic, which is unexplored to the best of the author's knowledge.

Therefore, the relevant basics of MAN and their differences from campus IT or Internet networks will be briefly identified. Then, the application of control engineering within data load control in communication networks is addressed. Finally, the different existing load control concepts listed above and their relevance for load distribution control for TSN MANs need to be reviewed more closely.

2.2.2 Manufacturing Automation Networks (MAN)

The field of automation is huge (Soldatos et al., 2019) as it is used in all kinds of applications such as power plants, chemical process industry, food and beverage industry, packaging industry and other industrial productions or manufacturing like vehicle production, for example, to name just a few. This research project focuses on manufacturing automation networks as they typically have more demanding communication requirements regarding speed and latency than other automation areas. These results are expected to be applicable to other fields of automation.

The general Ethernet network technology (Tanenbaum et al., 2021) has over the last decades produced two main principles for relaying data packets (Section 2.2.1):

- Data packet relaying by *routing* is based on layer 3 techniques. For example, the most popular and widely used network communication protocol within Ethernet networks is the Internet Protocol (IP), which uses IP-addresses as routing information.
- Data packet relaying by *switching* is based on layer 2 techniques. For Ethernet networks the so-called destination MAC-Address is usually used to forward data packets to the recipient.

For the MAN in focus within this thesis, Layer 2 networks are the most important, as they are mostly used in MAN setups at automation cells at the field level and controller level and automation cell interconnection areas, as depicted in Figure 1.1. However, layer 3 communication is also sometimes applied in automation, but mainly in hierarchical higher-level networks connecting larger automation areas within a factory or plant (IEC/IEEE 60802, 2018).

Another important property of a communication network is its topology, that is, the structure of how single network participants are connected. Several basic network topologies are shown in Figure 2.2.

Early legacy automation networks without redundant connections were preferentially set up in line or star topologies. For reasons of fault tolerance, media-redundant networks, that is, networks providing at least two paths between each node, are increasingly applied. To achieve redundant connections with a minimum wiring effort, the ring topology and redundantly coupled ring topology have become the prevalent topologies in redundant MAN (IEC/IEEE 60802, 2018). The transition from line or star topology to ring and redundantly coupled ring topologies, offers an alternative communication path that allows load distribution solutions.

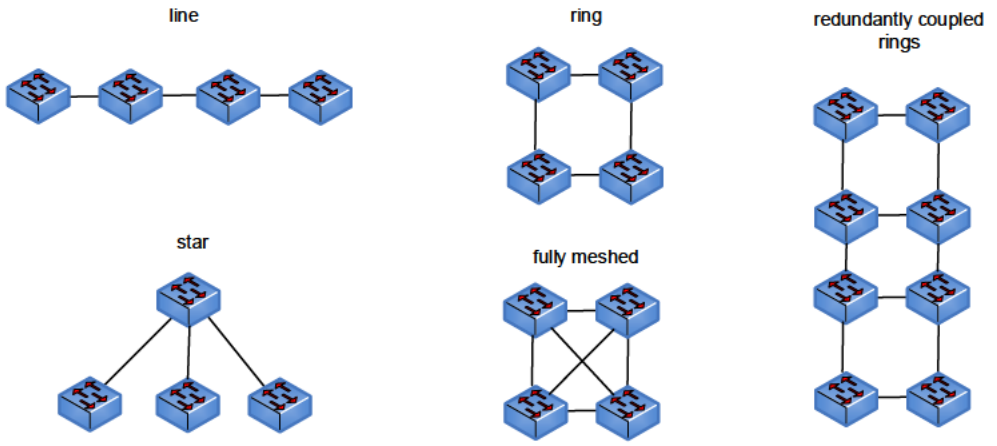


Figure 2.2: Basic network topologies

The communication objects exchanged between network participants in the form of data frames as a series of bytes depend on the communication protocols used in the different communication layers. Contemporary general examples of protocol representatives in OSI Layer 3 and above are protocols such as Http, Ftp, TCP, UDP, and IP. In this research the focus is on control data (CD) which typically have proprietary formats of proprietary protocols that do not follow the format definitions of the aforementioned protocols. Typical examples are EtherNet/IP (ODVA, 2023) or PROFINET (IEC 61158-5-10, 2023).

As an example, a typical application in many industrial automation applications is the synchronous speed control of several drives (Xiao et al., 2005; Yao et al., 2021). The major advantage of this is the saving of a mechanical gear to synchronize various axes and a very flexible change in speed templates and speed ratios between the drives.

This is an important use case example for this research project because it demands the CD of fast application communication cycles that are not allowed to be reduced in throughput.

Synchronous speed controls can be found in applications such as:

- conveyer belts with several drives
- elevator platforms with drives on independent platform corners
- cylinders of printing machines

- cylinders of rolling mills
- stations of packaging machines
- assembly feeding chains of automotive production.

Such synchronous speed control automation setups are usually part of larger automation applications together with various other applications. Figure 1.1 sketches a typical larger automation communication network setup, as described in IEC/IEEE 60802 (2018) or IETF RFC 8578 (2019).

In the automation network shown in Figure 1.1, an upper automation controller ring bundles access to several subrings that are responsible for controlling certain parts of the automation setup. These parts can be machines or automation cells, such as different conveyer belts in a series of conveyer belts or different assembly chains in a series of assembly chains. The upper controllers in the controller level ring have access to the single subrings for tasks such as changing production templates in the subring controllers, synchronizing the subring devices, reading visualization data, or obtaining general diagnostic data. In recent years, the trend has been towards increasingly outsourcing the tasks of the controller level ring to virtual ACs (Beran et al., 2010; Soldatos et al., 2019) onto so-called edge automation devices (Mandić et al., 2022; Singh et al., 2022) or to cloud-based automation (Ranjan et al., 2016; Wollschlaeger et al., 2017). Edge or cloud devices typically contain several virtual ACs. Because cloud-based automation involves higher and unsecure path delays between a remote cloud and an automation cell, it is not suitable for faster manufacturing control applications, as they are the focus of this research and will not be explored here. Edge solutions are equivalent to local AC solutions because the paths are typically only slightly longer.

Within the subrings, ACs, such as a PLC or MC, exchange process data with single peripheral devices such as sensors, actuators, digital/analog inputs, and outputs (I/O), or drives for the motors that drive the mechanical parts of the automation application. Regarding the data flow direction, sensors such as temperature, pressure, or humidity sensors provide data from sensor in the direction of the AC. In contrast, devices such as linear actuators, open-loop controlled motors, or valves cause data flow in the other direction from AC to the actuator. Drives,

closed-loop controlled motors, or completely decentralised peripheral automation stations for I/O typically cause bidirectional data flow. For this research project, both the field-level subrings and the controller-level rings will be the focus.

In the case of edge devices, in contrast to cloud-based automation, the communication paths are still relatively short and typically in the same layer 2 domain, as they usually installed directly in the production hall. Therefore, as Mandić et al. (2022) and Soldatos et al. (2019) also correctly state, edge-based automation is just as suitable for latency-sensitive applications as dedicated controller-level ring hardware. Thus, the results of this study can be applied to both dedicated AC hardware setups and edge-based AC setups.

For the CD relevant to this research project, different requirements regarding the communication cycles of the process data between the controller and drives or sensors exist (IEC/IEEE 60802, 2018). This depends on the automation application. For example, in the case of conveyer belts for transport purposes, the communication cycle speed is rather relaxed and may be selected between 10 and 100 ms. For synchronisation for printing machines or packaging applications, the communication cycles often need to be around 1 to 10 ms. Very demanding applications such as packaging or labeling require communication cycles below 100 µs.

The amount of data to be exchanged between the subring controllers and devices in the subring is typically rather low (IEC/IEEE 60802, 2018). As a rule, these exchange frames with data sizes of less than 100 bytes, as information exchange is typically limited to the cyclic transmission of setpoints and actual values. Other data to be sporadically exchanged between the controller and field devices can be diagnosis data and parameterization data. Because of their sporadic character, these are also called “acylic” data (AD). AD for diagnosis or HMI stations is transported in larger frames, sometimes up to a maximum frame size of 1550 bytes (IEEE 802.1Q, 2022). They are also transported at higher data rates which can easily reach a few megabytes per second.

It is well known, and worked out by Wisniewski et al. (2009), among others, that the use of media redundant networks demands the application of media redundancy protocols. Alvarez Vadillo et al. (2019) classify them and provide an overview. Media redundancy protocols influence the possibility of network load control. Basically, a distinction must be made between so-called “seamless” protocols, which send data doubly over separated paths, and path-changing protocols. The former cannot be combined with network load control, as they already use both paths at the same time. The latter are traditionally spanning tree protocols such as RSTP or MSTP (IEEE 802.1Q, 2022) which are slow though with reconfiguration times typically around a few seconds. Researchers such as Yuen et al. (2011) have successfully attempted to accelerate RSTP and MSTP reconfiguration times by working with backup VLANs. However, the results imply a complicated network setup, which make it difficult to use. These problems have led to the development of dedicated redundancy protocols such as the MRP (IEC 62439-2, 2021; Wisniewski et al., 2009). MRP provides faster network reconfiguration times between 500 ms and 10 ms. However, MRP has only been used to completely change the path for all traffic and does not have the property of MSTP VLAN-based path separation, which is important for network load control. In certain automation applications, the seamless flow of cyclic communication data is crucial for the quality of the automation tasks. An interruption in the process data stream or a certain delay in the process data can lead to product failures or even automation plant damage. For seamless redundancy, where data are sent doubly over two available paths from source to sink, thereby providing zero reconfiguration time, the HSR protocol (IEC 62439-3, 2021) can be applied. However, its application also excludes network load control. In combination with streams, which follow dedicated assigned paths, the seamless redundancy protocol IEEE 802.1CB, also called “Frame Replication and Elimination for Reliability” (FRER) was defined by IEEE 802.1CB (2017). The types of applied media redundancy protocols are important for NLB. These must be chosen and configured carefully depending on the traffic types present in the network (Alvarez Vadillo et al., 2019; Kirrmann & Dzung, 2006; Wisniewski et al., 2009).

However, this task is currently not reflected in the existing research in combination with NLB for TSN MAN and must also be addressed in this study.

As Lo Bello and Steiner (2019) outline, the desire of many users to transmit deterministic CD and non-real-time traffic (NRT) in parallel via the same data path has led to the foundation of the task group IEEE 802.1Q TSN TG (2022) to develop so-called converged networks, which finally got the name time-sensitive networks (TSN). TSN standards provide new traffic shaping and scheduling mechanisms and are also the basis for the latest modernisation of industrial automation networks.

2.2.3 Time Sensitive Networks (TSN) as MAN

Lo Bello and Steiner (2019), and Finn (2018), provide a comprehensive overview of TSN. These are networks designed in accordance with one or more of a set of new network standards (IEEE 802.1Q TSN TG, 2022) which essentially introduce the capability of delivering data within a guaranteed minimum time from the data source to the data sink. This property is known as “deterministic” data transport. A further essential achievement and advantage of TSN is that it provides the possibility of the coexistence of non-deterministic network traffic together with deterministic network traffic on the same medium. This is often referred to as a converged network. Converged networks are a major improvement in MAN to save separate networks for deterministic and non-deterministic data traffic for an application.

TSN functionality is shaped within the TSN Task Group (TG) as part of the IEEE 802.1 Working Group (WG). They defined a series of standard amendments, supplementing or building on the central standard for Local and Metropolitan Area Networks (IEEE 802.1Q, 2022). Farkas et al. (2018) provided a more detailed overview of these TSN standards and of further reading on this topic.

The TSN standards define a variety of traffic shapers and schedulers to achieve an optimal data transfer for CD. These are the Strict Priority Queuing (SPQ), or Static Priority Queuing as it is also called, Credit Based Shaper (CBS), Enhancements for Scheduled Traffic (EST), Cyclic Queuing and Forwarding (CQF), and Asynchronous Traffic Shaper (ATS). Further TSN-specific functionality are Frame Preemption (FP),

Scheduled Transmission (ST), and Stream Reservation (SR). Summarising the work of Lo Bello and Steiner (2019), and Finn (2018), the shapers, schedulers, and TSN-functions have their individual effect on data traffic transport:

The **SPQ** is known from classical Ethernet switch ASICs and is defined in IEEE 802.1Q (2022). SPQ is a favoured common system for assigning different Quality of Service (QoS) properties to various traffic classes in general Layer 2 networks and in automation communication networks. It is also used in TSN, mostly in combination with stream or bandwidth resource reservation (IEEE 802.1Qcc, 2018; IEEE 802.1Qdd, 2023). Grigorjew et al. (2021) show that thereby a deterministic TSN can be achieved, although SPQ is not a traffic shaper or scheduler in the literal sense. A contemporary example is the "PROFINET (PN) over TSN" field bus (IEC 61158-5-10, 2023; IEC 61158-6-10, 2023) for 1 Gbit/s and 2,5 Gbit/s which combines SPQ with express forwarding and scheduled traffic (EST, see further down for details) for CD and Non-CD data. It can also use SPQ in combination with synchronised traffic injection as an alternative to other methods of shaping or scheduling traffic. SPQ provides one egress queue for each or collections of the eight QoS frame priorities (IEEE 802.1Q, 2022). For CD, it is common to use the highest or, in cases where network control frames shall have a higher priority, the second highest priority. For the non-CD the next lowest priority is used.

SPQ is expected to be usable in connection with load distribution control (LDC) concepts for MAN and needs to be involved in the analysis and evaluations provided by this thesis.

The **CBS** was introduced by IEEE 802.1Qav (2009) and IEEE 802.1Qcc (2018) for transferring audio/video (AV) data without bursts and congestion. Zhao et al. (2022) provided extensive performance data of the CBS and demonstrated its suitability in AV applications to achieve a continuous data stream and avoid bursts. However, this advantage in AV is a disadvantage in automation applications where the challenge is to maintain the CD burst along a network path. The main feature of the CBS is that it stretches bursts of data to achieve a continuous flow of the stream. Therefore, it is of low interest for industrial automation CDs with fast

communication cycles, since CDs are intentionally sent in bursts at the beginning of a new application cycle from an automation controller. Consequently, it is not considered here for automation networks application and analysis.

The **EST**, defined by IEEE 802.1Qbv (2015), is also known as time-aware shaper (TAS). It assigns gating windows to traffic classes. The send queue of each traffic class is thereby emptied at a defined time slot, that is, the gating window or gate open window, which is repeated every network cycle or gating cycle. The expressions gating cycle, network gating cycle, or short network cycle are used synonymously in the literature and standardisation. In addition to other researchers, Craciunas et al. (2016) have made investigations as to the performance of EST. EST can be used to have synchronised gating windows in all bridges of the TSN domain, with no other data interfering with the transmission during the gating window. This guarantees a free path for data traffic and minimum network latency through the complete synchronised EST network domain. If in addition a synchronised talker sends at the start of the gating window, minimum network latencies can be achieved. Craciunas et al. identified and analysed key parameters affecting the deterministic behaviour for centralized scheduling of an EST. They also proposed optimisations to reduce the scheduling calculation time by introducing heuristics. Vlk et al. (2020) enhanced the schedulability and throughput of EST by adding a queue order check but required special hardware additions to realise it. However, the application of specialized hardware implies a high development effort and thus low market acceptance. Still, the already low EST latencies are a sufficient reason to include EST in this study for LDC in TSN MAN.

The **CQF** traffic shaper, defined by IEEE 802.1Qch (2019), also follows a network-domain-global network cycle. It stores the ingress traffic in one network cycle, which it then forwards during the next network cycle. Through this method, a certain amount of data traffic is handed from bridge to bridge taking one hop per network cycle. Finn (2019) shows that with CQF, limited latency can be guaranteed which depends on the maximum number of hops through the CQF network domain. In this case, the latency per hop is identical to that of the network cycle.

The amount of admissible data per cycle depends on the configuration of the cycle length and can be restricted by reservation and ingress limiting. Finn also outlines, that windows for data of further traffic classes to be transported in parallel, raise the necessary gating cycle and thereby the latency accordingly. Huang et al. (2022) addressed the relatively high jitter of CQF and proposed a combination of CQF with time triggered transmission, as known from the EST.

The suitability and characteristics of the basic CQF must be included in the analysis and evaluation of LDC for TSN MAN.

The **ATS**, defined by IEEE 802.1Qcr (2020), provides additional, shaped egress queues that feed the existing classical egress queue structure of SPQ. The processing chain for a stream with ATS consists of Per-Stream Filtering and Policing (PSFP), shapers, egress queues, transmission selection and gate control. An Internal Priority Value (IPV) can be assigned to each traffic class within a bridge, independent of and without influencing the frame`s tagged priority, allowing dedicated prioritized frame handling per hop. A detailed analysis of the ATS properties was provided among others by Zhou et al. (2018). They show that the ATS does not depend on synchronous bridges or synchronous communication and offers bounded latency for lower performance control data such as non-isochronous CD (NI-CD). The ATS shaper mechanism works as a Token Bucket traffic shaper, that is, it allows bursts to be limited to configurable bursts. Nasrallah (2019) performed comparison with EST and found that EST is better suited for cyclic traffic such as CD for automation purposes, whereas the ATS has more advantages with sporadic, lower priority traffic. However, as the ATS is one of the main TSN shapers, the analysis and evaluation of its capabilities in connection with LDC for industrial automation is mandatory.

FP according to IEEE 802.1Qbu (2015) is another TSN feature where streams are classified as either express traffic or preemptable traffic. Express traffic can interrupt the transmission of a preemptable frame and thus overtake preemptable frames. After the express frame is transmitted, the preemptable frame transmission is resumed. The frame preemption feature according to IEEE 802.1Qbu is defined for the MAC Layer and strongly correlates with the definitions

within IEEE 802.3br (2016), Interspersing Express Traffic (IET) for the Physical Layer. Thiele and Ernst (2016) performed a worst-case performance analysis of TSN with FP and demonstrated that it can accelerate higher priority traffic. Preemption can basically be applied in Strict Priority scheduling environments but can in principle also be combined with EST, CQF and ATS shapers. Logically, only one traffic class can be classified as express traffic without spoiling the intention of preemption. A performance comparison of FP versus EST was provided by Hellmanns et al. (2020). It demonstrates the advantages of FP in zero planning and synchronisation requirements. However, it also shows the disadvantage of FP, in that, with increasing use, the latency reduction degrades towards pure SPQ. Against this background, the proposal of Nikolic et al. (2020) for a multi-level FP to realise further express traffic classes, seems rather questionable. Nevertheless, the improvements achieved with the moderate application of FP in combination with SPQ for time-critical data make it an interesting feature worthy of inclusion in the investigations of this thesis.

ST of data frames is another method that arose with the creation of the TSN framework. The required scheduling algorithms were the focus of several research teams (Ojewale & Yomsi, 2020; Y. Song et al., 2021; Yang et al., 2021). The background to their work is the idea that the transmission time of each frame can be planned in such a way that optimal utilization of the entire network is achieved with minimal path latencies and optimal bandwidth utilization. They propose their individually found solutions for heuristic algorithms to achieve this. These solution always require a central instance, typically a central network controller (CNC) as defined by, for example, IEEE 802.1Qcc (2018). This calculates the optimised transmission points in time for each frame and station, at least to a certain extent. However, as Craciunas et al. (2016) and Falk et al. (2019) explain, the problem with ST is that it requires extensive calculations for an optimised schedule for all traffic. As this thesis focuses on real dynamic load distribution though, such methods, which require pre-network-startup calculations or complete reconfiguration at runtime, are only of secondary importance (see Subsection 5.3.7 for more reasoning on this).

SR is another crucial feature that can be used in the TSN domain for resource reservation. The RSVP (IETF RFC 2205, 1997) is an early representative of a resource reservation protocol for the transport layer 3. It is primarily used in multimedia applications (Braun, 1997). SR is represented in the IEEE TSN framework by IEEE 802.1Qcc (2018) as the Multiple Stream Reservation Protocol (MSRP), and by IEEE 802.1Qdd (2023) as the Resource Allocation Protocol (RAP). Grigorjew et al. (2021) investigate SR in combination with automation networks in its original application as overload protection for the network, where excessing streams will not obtain a bandwidth reservation in the bridges. They showed that with SR and admission control a bounded latency can be achieved with standard Ethernet switches which contain no further TSN shapers or schedulers. Thus, latency guarantees can be provided for certain stream classes. To protect against congestion, that is, against talkers that exceed their reserved bandwidth, ingress limiters as defined by IEEE 802.1Qci (2016) can be used as supplementary protection. However, which also from the work of Grigorjew et al. emerges, the reservation process needs time, which in combination with a dynamic shifting of traffic from one path to the other is disruptive. These implications of SR for LDC must be included in the investigations of this thesis.

A deeper insight into the timing behavior of each shaper and scheduler was provided by Falk et al. (2019), based on OMNet++ network simulations. Zhao et al. (2022) performed a quantitative performance comparison of various TSN shapers. Roughly summarized, their extensive studies show that SPQ has an advantage for traffic classes with high priority. A variety of traffic priorities can be optimally controlled with CBS. The ATS is the best choice for fair scheduling of lower priority traffic. The EST (or TAS) offers ultra-low latency and jitter and is suitable for all priorities and traffic classes with dedicated gate control windows but is difficult to manage in combination with larger networks. The focus of this research is on the transport of higher-priority CD when they are not transferred seamlessly. Thus, according to Falk et al. and Zhao et al., it can be expected that the application of SPQ and EST will be beneficial.

In summary, it can be stated that these traffic shapers and schedulers have, owing to their different influences on path delays, a different influence on data flow control or load distribution control. Therefore, a deeper analysis of SPQ, EST, CQF, ATS, FP and SR must be included in the investigations in this thesis.

2.3 Methods for Load Control in Communication Networks

Existing control methods for a variety of control problems in general dynamic systems have also been applied to communication networks. With the availability of multiple paths in general communication networks, the research community has developed load control concepts not only for congestion control but also for the distribution of loads (Section 2.2.1). The most important load control methods are introduced and investigated hereunder.

2.3.1 Load Reduction Strategies

Communication paths within communication networks have limitations in terms of the maximum transferable data amount per time, which in communication networks terminology is also called the maximum "bandwidth" of a communication path.

To protect a communication path from overload during runtime of the network, which could result in the loss of data, it is necessary to either reduce the amount of data to be transferred, or to distribute the data on additional communication paths. Thus, as is clear from Section 1.1 and the literature listed in Subsection 2.2.1, there are two basic methods. The first is **"congestion control"** (Kanagarathinam et al., 2020; Kasoro et al., 2021; Katabi et al., 2002), the second is usually named either **"load balancing", "load distribution", or also "load sharing"** (Jahde et al., 2021; Kandula et al., 2007; Lemeshko et al., 2013).

Congestion control can be either open- or closed-loop. As Wu and Mark (1993) outline, open-loop congestion control has no feedback for the actual load and is thus "blindly" or "obliviously" reducing load. Contrarily, to achieve control results of higher stability, Katabi et al. (2002) apply closed-loop control. Therefore, a

control system must be established that measures the actual amount of data on one communication path and initiates corrective actions to reduce the amount of data being passed onto this communication path.

The system output to be controlled and the system input in both congestion control systems and in load balancing control systems, is the network data path data rate per time termed network path bandwidth use or load.

For load balancing, the control system provides an evenly distributed load on several possible communication paths.

Another way to achieve a balanced load is to properly plan and configure the traffic distribution before network startup as it has been applied in various studies (Lopez et al., 2010; Santos et al., 2009). This method is called “Traffic Engineering (TE)” and should always be part of the network **setup.** However, because this research focuses on dynamic load distribution, TE methods are of secondary importance.

Sometimes the procedure of dynamic load distribution is also called “Dynamic TE” or “Adaptive TE” (Elwalid et al., 2002; Wang et al., 2006).

In this thesis, the focus is on load balancing, or more generally, on load distribution rather than on congestion control. This is because CD and particularly isochronous CD (I-CD) are subject to a bounded admissible latency, that is, traffic throughput must not be reduced significantly. However, as it emerges from the work of Puqi Perry and Tai (1999), some methods of congestion control, such as the Token Bucket Shaper, are important as they are used as part of the CBS and ATS.

The goal of load balancing or load distribution systems in communication networks is to achieve a controlled and as even as possible distributed path bandwidth use of all available data paths from data sources to data sinks, as outlined among others by Antic et al. (2010) or Prabhavat et al. (2012).

When considering layer 3 routed multipaths networks, the three main types of traffic control concepts (Wang et al., 2006), as introduced in Section 1.1, oblivious routing (Kandula et al., 2005) (Räcke, 2009), traffic control using predicted traffic demands (Otoshi et al., 2015; Wang et al., 2006) , and adaptive or dynamic routing

are visible. As also outlined in Section 1.1, only the latter is important for this study and to be reviewed closer.

Adaptive or dynamic routing is based on routing decisions based on network load metrics. There are various possibilities for obtaining these metrics. Kandula et al. (2005) send test frames onto single paths, where each router inserts its current bandwidth use. The test frames were sent back by the last router at the edge of the routing domain. Another possibility is to poll the port load of each router along the path (Elwalid et al., 2002). Various control concepts, such as Common-case Optimization with Penalty Envelope (COPE) by Wang et al. (2006) or Smith-Predictor by Mascolo (2000), have been used to achieve a balanced load on the OSI Layer 3 ISP networks. The adaptive or dynamic routing concept is identical to closed loop network load control (Subsection 2.3.2) and has been extensively investigated by research, as further elaborated in the following sections. Because of its better reaction time on load changes than open-loop control, it is a relevant method for this research project within TSN MANs.

2.3.2 Closed-loop Dynamic Load Control

A desired quick and appropriate response to changes in a dynamic system, such as load changes in a MAN, can only be achieved with a closed-loop control (Goodwin et al., 2001).

In order to classify the types of control systems applied in the literature, it is necessary to distinguish between their elements. Goodwin et al. (2001) define a control system as follows:

Figure 2.3 shows the main parts of a typical dynamic system in combination with a controlling entity that is used to control the system output.

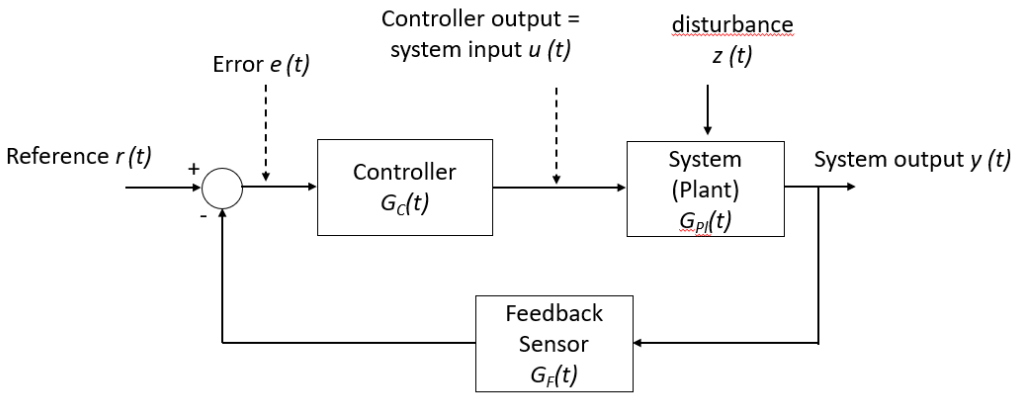


Figure 2.3: General control system

Accordingly, according to Godwin et al., systems can be abstracted by dividing them into three main parts:

1. The system to be controlled, which is also called “the plant” in control engineering terminology, with its output *y(t)* to be controlled. In the case of this research for data traffic LDC, the system to be controlled is the communication network with its output of data per time, that is, the bandwidth as a scalar value.

2. A sensor module feeds back the system output value to compare it to a reference value *r(t)* representing the desired output value. The sensor is represented by data load measurement hardware or software in the network devices, that is, routers, bridges, or bridged end stations. The reference value for the network path is again a bandwidth as a scalar value.

3. The controller controls the system input based on the difference *e(t)* between the reference and the feedback of the output: $e(t) = r(t) - y(t)$. In TSN MAN the controller is part of the end devices or bridges that control the data ingress rate.

Goodwin et al. also state that, associated with the types of systems to which a control system is applied to, the control theory can be roughly divided into two main areas. The “Linear Control Theory”, which deals with linear control systems and the "Non-linear Control Theory", which deals with non-linear control systems. Since the data throughput on a communication path has to be measured

continuously over a period of time, for example as a moving average, it changes linearly and the LDC for data traffic can be classified as a linear control system.

The literature offers a variety of different closed-loop load control solutions. Katabi et al. (2002) further developed the congestion control algorithms for TCP by introducing eXtended Control Protocol (XCP) with analog feedback. This was obtained by measuring the round-trip time (rtt) of a TCP flow. It allows conclusions to be drawn about the load on the path. They used a linear controller and introduced stability analysis according to Nyquist (Normey-Rico & Camacho, 2007). The linear controller, such as a proportional-integral-derivative (PID) controller, is also applicable for distribution control and flow control of MAN CD traffic type. However, the load measurement method using the TCP rtt is not applicable, because CD does not provide acknowledgement frames back to the sender.

Elwalid et al. (2002) presented with “MPLS Adaptive Traffic Engineering (MATE)” a minimalistic adaptive TE approach in multi-protocol label switching (MPLS) networks, which also manages without load measurements along the path. It is based on end station feedback by measuring the delay experienced by a probe packet along a path from ingress to egress node. The distribution controller is implemented by using a dedicated algorithm. However, as with Katabi`s et al. XCP, this method is only suitable to make an end-to-end load statement for de-loading complete paths from a talker to a listener. It is no solution to resolve load maximums at certain links as it is the goal of this research on MAN LDC.

Another closed-loop LDC in the form of adaptive TE was provided by Kandula et al. (2005). They developed “Traffic Engineering eXtended Control Protocol (TeXCP)” based on Katabi`s et al. XCP, where each router along a path updates a probe packet with its port utilization if it is greater than the one contained in the probe packet. The egress node sends the probe back to the ingress node which applies a dedicated balancing algorithm to distribute the load on the paths. However, as with MATE, this approach does not solve link load maximums along the single links on the paths. Furthermore, in an automation ring, each listener would report back different results.

The review on closed-loop LDC approaches shows that they have been applied in different network application domains, with different load feedback methods, and using different types of controllers.

Summarising the load feedback options, there are three:

1. The listener end-station sends the load maximum back to the talker. This is conveyed by a probe packet sent by the talker, which is updated by the nodes along a path.
2. The listener end-station sends back to the talker the path delay experienced by probe packet sent by the talker.
3. Each node in the network provides a port load value that can be polled by a talker.

The first method may work well with a limited number of talker and listener nodes, such as edge routers, and thus a limited number of probe packets. However, in larger MANs with hundreds of automation end stations, there will be significant additional traffic. The second method is regarded as imprecise because delays along a path can also be caused by sporadic interfering traffic, that is, it is not necessarily proportional to the load along a path. The third method is considered the best as talkers can poll each node and build their own complete network load view.

A more complex situation arises with the controller types used. Researchers applied all important and proven types to solve LDC in the different network types.

2.3.3 Controller Types

As is clear from Goodwin et al. (2001), a key distinguishing feature of closed-loop LDC is the type of distribution controller used. Its selection depends heavily on the type of network. The literature provides LDC solutions that include a variety of distribution controller solutions that have their own strengths and weaknesses.

The variety of load controller types used in network load balancing or network load distribution control essentially includes the following:

1. Linear control (Kandula et al., 2007; Wang et al., 2006),

2. Stochastic control (Neely et al., 2008),
3. Fuzzy control (Pompili & Priscoli, 2008; Talaat et al., 2019; Wang & Hung, 2012),
4. Smith control or Model Predictive Control (Mascolo, 2000; Quang et al., 2020),
5. Ant colony control (Mohammadnia et al., 2016; Zhang & Zhang, 2010),
6. Neural network control (Talaat et al., 2019; Wang & Hung, 2012),
7. Dedicated algorithm control (Elwalid et al., 2002; Farahmand et al., 2005)
8. Control by Artificial Intelligence (AI) or Machine Learning (ML) (Anna Victoria Oikawa et al., 2020; Todorov et al., 2020),

These control methods differ as follows.

Linear control is the traditional method for controlling a linear system (Goodwin et al., 2001). In its simplest form, the measured system output is fed back and is weighted by a proportional factor using a proportional controller. Further controller improvements led to the addition of an integration element and a derivative element, that is, to a proportional-integral-derivative (PID) controller, to achieve an optimum system output in terms of settling time, stability, and output constancy. Katabi et al. (2002) used linear network load control for efficiency and fairness controllers in their proposal of the eXplicit Control Protocol (XCP) to improve the TCP congestion control mechanisms. It can be used for both distribution control and subsequent flow control in an LDC. Its advantage is the low resource consumption of computing power and memory. However, it should be noted, that the actual optimum sizes of the PID parameters K_P, K_I *and* K_D depend on the characteristics of the network and must be adjusted individually for each network. This process is known as “tuning” of the controller (Normey-Rico & Camacho, 2007). This means that a controller must adjust this set of parameters for changes in the network that imply path length or delay changes. Another example of linear control, intended for high-delay networks is the Smith Predictor application by De Cicco et al. (2011). This must be applied when the path delays are high, compared to the time constants of the load measurements mean value calculation.

Linear control needs to be evaluated for its beneficial application in this study.

Stochastic network load control refers to the control or optimisation of a stochastic network, that is, traffic occurs in a stochastically distributed manner, rather than to the method of a stochastic controller itself (Åström, 2012). Stochastic network control assumes that exogenous data arrive at the network through random processes. This implies that the data rate and arrival intervals are randomly or stochastically distributed. In principle, it uses the average data arrival rate by building long-term mean values, which improve with longer integration time intervals. The crucial difference from the traditional linear PID control principles is the applied time interval. Stochastic network control applies longer time intervals for the assignment of the average data rate, as no known pattern can be assumed. Neely et al. (2008) emphasise, that the cost for a high-quality stochastic control for load control in communication networks is a higher end-to-end delay for the transport of the load values. Stochastic control can be applied to all core control problems such as continuous or discrete time, linear, non-linear, or predictive control concepts.

However, since the MAN CD is typically not distributed stochastically, but is transmitted cyclically, it is obvious that this type of control does not have to be applied to the networks of this research.

With the **fuzzy control** method (Matía et al., 2014), general plant parameters to be controlled are collected and fuzzified before processed by the fuzzy controller. Fuzzification means that the precise values of the input parameters are classified into certain ranges. The same method is applied to the controller output parameters. Such ranges could include classifications such as “very low,” “low,” “medium,” “high,” and “very high” when classifying for example the load on a certain network path. The fuzzy controller then applies a set of fuzzy rules on the input parameters, such as “If load is low, then raise output to high.” The controller compares all fuzzy rules and defuzzifies the results by deriving a precise result for the control output. The best-suited fuzzy rules, appropriate input parameter ranges, and output ranges are typically empirically obtained during the design of the control circuit. It is widely accepted and also pointed out by Pompili and Priscoli (2008) and Wang and Hung (2012) that fuzzy control is particularly

advantageous when the system to be controlled is either fairly complex, its behaviour is difficult to describe mathematically, or both.

However, these system properties typically do not apply to the MAN (Subsection 2.2.2), which are the focus of this research. Therefore, fuzzy control is obviously not the control method of first choice in this context.

Bolla et al. (1998) presented an early example of research on **neural network control** within mobile network load distribution, to realise dynamic bandwidth allocation. They used it to optimise bandwidth usage for combining isochronous traffic with asynchronous, statistically distributed traffic. For neural network control, according to Hagan et al. (2002), a neural network is used either as a function approximator or as a neural controller. The neural function approximator is used to approximate an unknown function, that is, the system to be controlled. In the latter case, different types of neural network control methods are possible when using the neural network as neural controller. Examples include Neural Model Predictive Control (Patan, 2015) and Model Reference Control (Patino & Liu, 2000). Let's consider Neural Model Predictive Control as an example. Neural Model Predictive Control, similar to Linear Model Predictive Control, is based on a predictive control approach in which the system under control is approximated by a neural network instead of being described mathematically. Hagan et al. (2002) explained the principle of neural network predictive control as depicted in Figure 2.4.

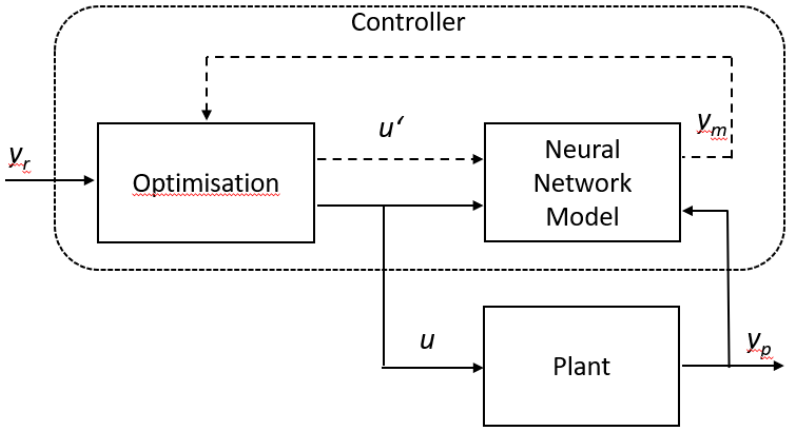


Figure 2.4: Neural network predictive control (Hagan et al., 2002)

Control was achieved by training the neural network to serve as a model of a complex non-linear real plant. This system identification stage is typical for neural control, including the aforementioned neural control methods. For this purpose, the plant output y_p together with the plant input u are fed into the neural network model and the plant output y_p is compared with the model's output y_m. The error between y_p and y_m is used to train the learning algorithm to improve the neural network model. The neural network model predicts the plant responses over a specified time horizon. Using the model, the numerical optimisation process of the controller determines the control signal u and tentative control signal u'. According to (Hagan et al., 2002) the application of Neural Network Model Predictive Control is advantageous with non-linear and/or complex plant properties that are difficult to describe mathematically. It is an alternative to the fuzzy control method for such networks. Additionally, a combination of both control methods can be an alternative (Talaat et al., 2019; Wang & Hung, 2012).

As already stated for the fuzzy control algorithm above and outlined later in Chapter 4 , the limitations of non-linearity or mathematical complexity are not really present for MAN compared to other networks. Thus, neural network control for LDCs within MAN seems unnecessary.

Dedicated algorithms for network load control have been used for both congestion control and distribution control. Such an algorithm typically follows empirically obtained rules for increasing or decreasing the loads. TCP congestion

control (IETF RFC 2001, 1997) is a very well-known and widely used representative of dedicated algorithm congestion control on the Internet. This is reviewed in more detail in Section 2.4.2. Various derivatives of this algorithm, such as TCP Reno (Hasegawa et al., 2000; IETF RFC 6582, 2012), based on different load increase and decrease strategies and decision criteria have been developed over the years. However, these were early attempts of closed-loop load control with simple control algorithms similar to On-Off-Controllers. These still failed to provide the possibility of closed-loop stability considerations. Meanwhile, other scientists such as Elwalid et al. (2002), with the multipath adaptive traffic engineering (MATE) approach, achieved improved control stability by applying higher-resolution analogous feedbacks. Farahmand et al. (2005) provided another example of a dedicated algorithm applied on optical burst links. They use a proportional load feedback and imply a model of the network characteristics thus achieving a complete closed-loop system. It achieves an improved network utilization without the occurrence of congestion.

In summary, it must be concluded that the dedicated algorithms developed by early LDC researchers eventually led to the application of more professional control engineering algorithms such as traditional PID controllers. Therefore, the development of a new dedicated control algorithm, different from those that have been widely applied and tested in previous research, as outlined in this section, does not seem promising.

Another possibility of controlling the network load is **control by Artificial Intelligence (AI)**, that is, **Machine Learning (ML)** (Russell & Norvig, 2021). This is known as **Machine Learning Control (MLC)** (Duriez et al., 2017). MLC is the correct selection when it comes to controlling complex nonlinear systems, where the traditional linear control theory is not applicable or is only applicable with a very high effort. For example, Duriez et al. (2017) used MLC for fluid turbulence control, which is known as one of the most demanding non-linear control problems, and defined MLC as follows. ML uses data to generate a system model. With more data, the models should improve and be capable of handling data constellations not yet seen before.

ML is typically classified into three types: supervised, unsupervised, and reinforcement learning (Müller & Guido, 2017; Russell & Norvig, 2021). Supervised learning uses pairs of labelled input and output training data to train a system. Yan Song et al. (2021) use supervised learning for routing and scheduling for simultaneous transmission of diverse data streams for NLB. They use the K-nearest neighbour (KNN) algorithm. KNN assigns data set members, in this case transmitted streams, to classes of data, in this case traffic classes, depending on their properties` relative distance to the class properties mean. This method can be used for classification and for regression problems. Unsupervised learning dispenses with the use of labelled output data and the system uses self-organised learning usually by using probability densities within the input and output relation. However, this is the more demanding method which implies higher processing power and memory demand in a controller. Reinforcement learning also lacks a learning phase. It works with a cumulative reward system to decide between successful and less successful control decisions.

Thus, MLC wraps ML algorithms around a complex system to learn an effective control law. This is particularly useful for systems in which it is difficult or impossible to develop a mathematical control law for a mathematical model of the system. A machine learning algorithm is chosen to find the best control law through training procedures with data from a training phase or from a simulation. Here, the transition from rule driven ML algorithms to data driven ML algorithms is important. According to Duriez et al. (2017), well-proven algorithms are evolutionary algorithms and, in here particular, the genetic algorithms for discovering control laws in high-dimensional search spaces. Duriez et al. described the control loop for the MLC, as depicted in Figure 2.5.

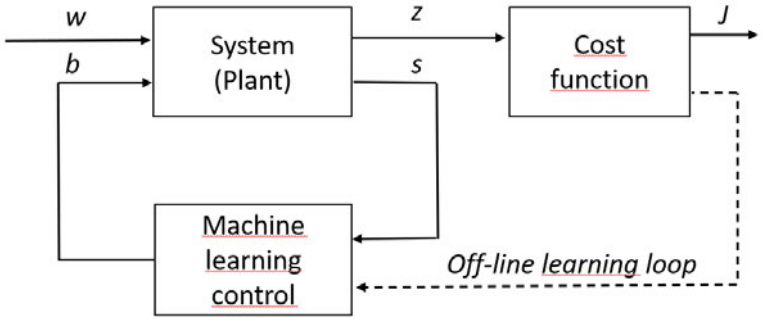


Figure 2.5: MLC control loop (Duriez et al., 2017)

The main difference between MLC and other methods listed above is, that no model, neither a mathematical model nor a neural network model, of the system is needed as it is with the traditional control methods. The MLC directly learns control laws without requiring a system model. According to Duriez et al., genetic algorithms can be used as optimisation algorithms within MLC. They use components such as chromosomes, selection, recombination, or mutation to create a genetic variety to explore a solution space. A chromosome is equivalent to a set of parameters (genes), that is, a possible solution vector of a certain quality. Quality is assigned by the cost function. The goal of the algorithm is to find a chromosome, that is, a solution vector that satisfies the quality requirement of a limited value of the cost function. For example, MLC has been used in research together with software defined networks (SDN) (Jahde et al., 2021; Todorov et al., 2020) which are well-suited because of the configuration flexibility of SDN bridges.

MLC has, according to Mowei et al. (2018), numerous processing steps, including problem formulation, data collection, data analysis, model construction, deployment and inference, and model validation. Thereby it is a very demanding control method regarding memory consumption and CPU calculation effort. Given that LDC in MANs is a linear control problem with limited mathematical complexity, the application of a demanding MLC control algorithm seems disproportionately high compared to a few tens of the lines of code for a PID controller.

Ant Colony Optimisation (ACO) control **(ACOC)** is another network load balancing technique, also typically classified as an AI algorithm. This is a metaheuristic algorithm inspired by the behaviour of a real ant colony (Mohammadnia et al., 2016; Zhang & Zhang, 2010). Zhang and Zhang (2010) used it for cloud computing networks and Mohammadnia et al. (2016) provided an ACO based solution for vehicular ad-hoc networks (VANETs). With these approaches, packets are given their best routing path decision in each individual node based on an accumulated "pheromone" value. The ants are represented by request and response data packets called agents which are sent from talker to the listener and sent back along the paths found. These packets contain information about the source address, destination address, number of nodes crossed and other quality criteria such as path cost, accumulated path delay, or maximum load along the path, from which the pheromone values are formed. Due to the decentralised character of this control, no load feedback to a central distribution controller is necessary. One of its advantages is that it is flexible in terms of network size and participating number of nodes. This also makes it a potential candidate for MAN LDC, which can also experience network expansions during ongoing operations. However, it has the disadvantage of not being able to anticipate congested links along a path. A routing decision can therefore turn out to be the wrong decision on the next link. This is particularly likely in ring topologies with limited paths, since a routing decision will not be followed by alternate paths, as in a highly meshed network. For the LDC in MANs, which is the focus of this research, it is in principle suitable for distribution control. However, since the prevailing MAN topology is a ring with only two available paths, it cannot fully exploit its strengths here. In addition, ACOC is not suitable for the subsequent flow control. This is totally absent from an ACOC approach, meaning the load must be shifted incrementally, potentially leading to unwanted control oscillations.

2.4 Network Congestion Control

Network congestion typically results from sporadic traffic peaks. These can be caused, for example, by sporadically accumulated network access attempts or simply by overloaded networks due to increasing usage. Excessive network congestion can result in a high or total loss of service, known as the congestive collapse of a network.

For example, early important considerations on the subject of congestion control provided the theories of Kelly et al. (1998). They consciously differentiated between the possibilities of traffic reduction, in the case of congestion control, and re-routing in the case of load distribution. They analysed simple additive increase/multiplicative decrease algorithms, that is, dedicated control algorithms according to Subsection 2.3.3. These have been applied to congestion control and fairness control between services, such as is common in a TCP congestion control algorithm. They show that stability around an equilibrium point can be achieved for both controls. For fast MAN CD traffic, which must not be significantly delayed, however, the effect of congestion is fundamentally unsuitable and must be avoided. Congestion control is therefore not the focus of this study. It is only applicable to slower applications, where limited delay of the CD does not spoil the intent of the application. However, its principles are important for the TSN shapers CBS and ATS and must therefore be reviewed.

Control or avoidance of congestion is typically achieved by limiting the ingress rate of network traffic (Nasrallah et al., 2019). It can be regarded as a pre-stage for load balancing. Congestion control is typically limited to a single network path. Some studies have also used the expression “congestion control” in the context of multipath networks which is similar to load distribution or load balancing. However, this is an exception and was rather used at the beginning of research on load balancing.

Depending on the length of congestion, different counter measures are advisable (Jain, 1998). A very short-time overload can be compensated by adequate message buffering with sufficiently sized egress queues at the output ports of

network bridges or routers. Long-time congestion usually indicates that the network is undersized and requires a major upgrade. Mid-time congestion can usually be avoided by applying favourable additional network designs, such as:

1. Open-loop congestion control, or
2. Closed-loop congestion control.

2.4.1 Open-loop Congestion Control

As Nasrallah et al. (2019) have pointed out, open-loop congestion control attempts to avoid overload situations by stretching overload peaks or limiting the amount of egress data at the sender side and/or in the bridges. Stretching of traffic peaks can be achieved by applying a favourable sender or bridging traffic shaping algorithm, such as the Leaky Bucket traffic shaper or the Token Bucket traffic shaper (Tanenbaum et al., 2021). These basically follow the principle of buffering excess ingress network traffic, which is freed afterwards at the egress side at a constant rate over time. With that they obtain a homogenous traffic distribution over time. In addition to protecting against overload, a positive side effect of bucket-based shapers is that they safe queuing resources in subsequent bridges because they defuse traffic bursts (Falk et al., 2019).

The **Leaky Bucket algorithm** has been widely applied and investigated in research such as provided by Wu and Mark (1993). It is based on the analogy of a bucket with a leak as illustrated in Figure 2.6.

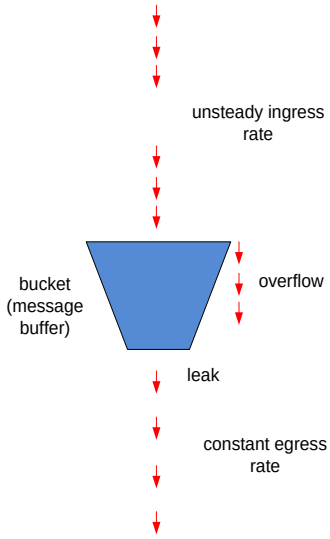


Figure 2.6: The leaky bucket principle

The symbolic bucket could be filled at an unsteady rate but is emptied at a constant rate via the leak. Excessive input results in overflow.

In this sense, transferred to data communication, as explained by Wu and Mark (1993), the ingress data frames are buffered in a message queue. This will be emptied at a constant rate via the output port of a network device such as a router or bridge. If the amount of ingress data over time exceeds the egress rate for a longer period of time, depending on the memory size of the queue, the queue will be filled sooner or later completely, and further frames will be dropped. Thus, short ingress peaks are tolerated and smoothed without data loss. In connection with this research, the leaky bucket principle is relevant because it is similar to the TSN CBS shaper (IEEE 802.1Qav, 2009). Furthermore, its ability to save queuing resources makes it an attractive add-on in combination with other shapers or schedulers, if the applications allow CD stretching.

Another bucket-based shaper is **the Token Bucket algorithm** (Ohnishi et al., 1988; Puqi Perry & Tai, 1999; Tanenbaum et al., 2021). It does not buffer frames as the leaky bucket algorithm does. Instead, it buffers tokens that are generated internally at a constant rate T_{Token} until an adjustable number of *n* tokens have filled the buffer (bucket). In this event the buffer is emptied, and *n* frames are sent out during a burst. Thus, the token bucket algorithm does not smooth out the ingress bursts of frames by sending frames at a constant rate. Instead, it smoothes

bursts of a large number of frames to bursts with a limited number of frames. Thus, it is better suited for CD than the Leaky Bucket algorithm. Figure 2.7 illustrates this principle.

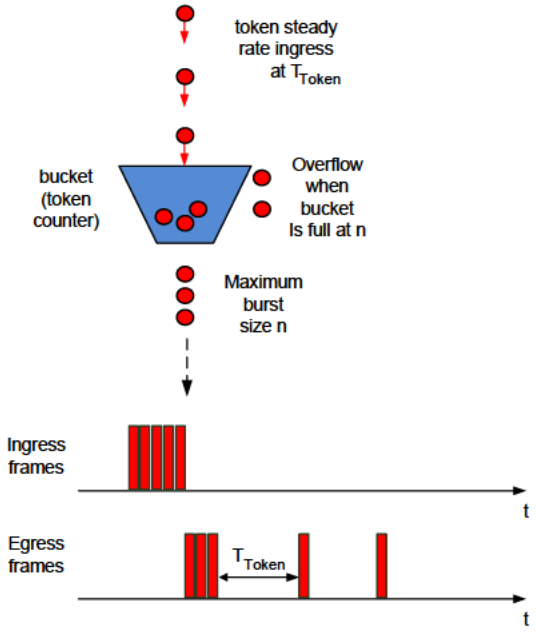


Figure 2.7: The token bucket principle (Tanenbaum et al., 2021)

With this implementation the token bucket algorithm has the following properties regarding the ingress-bursts of frames, such as CD:

- Longer gaps between bursts lead to an immediate transfer of the following burst, but to the limit of *n* frames (bucket size).
- Long bursts are cut into one burst of *n* frames, followed by a succession of frames at a rate of T_{Token-}

In addition to the prevailing token bucket concept of equaling one token with one frame, according to Tanenbaum et al., there is another form that equals a token with a number of bytes, which is advantageous in applications where higher amounts of data are transferred.

Compared with the leaky bucket algorithm, the token bucket algorithm has the advantage of avoiding loss of data frames. It also allows for faster reaction times from a sender`s point of view as bursts are allowed to propagate through the network, although of a limited size. Thus, it is much better suited for handling

congestions of CD as it still allows frame bursts which can be limited to the size of an ACs number of frames for one application cycle. Moreover, it is also important as it is also a component of the ATS scheduler (IEEE 802.1Qcr, 2020) of TSN and can thus form the properties of the TSN data path.

The **credit-based shaper algorithm** (IEEE 802.1Qav, 2009) is based on the work of Bensaou et al. (2001) which proposed the Credit-Based Fair Queueing (CBFQ) algorithm. It is suitable for distributing available bandwidth among different traffic classes on the same port. This is achieved by allocating send credit to data traffic during waiting times and withdrawing credit while sending data traffic. With increasing waiting time, a per-traffic queue credit parameter, constituting the send credit of a certain traffic class, is increased at a constant rate over time. This parameter is commonly denoted as idleSlope. The next frame of this traffic class waiting in the queue is sent with the reach of a certain amount of credit, represented by the hiCredit threshold parameter. The sending process decreases credit with the sendSlope rate. Sending is possible until the parameter lowCredit border is reached, provided that there are further frames in the queue. Figure 2.8 illustrates this credit-based shaper principle.

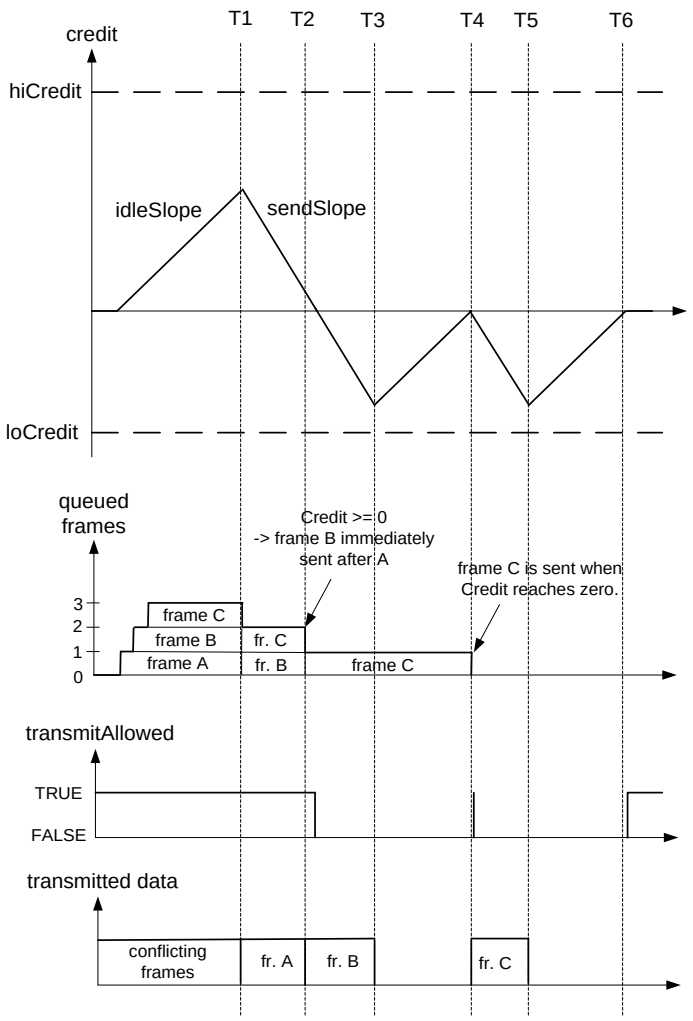


Figure 2.8: The credit-based shaper algorithm

As mentioned above, CBS can also be applied in TSN MAN in combination with slower CD cycles, allowing CD stretching without consequences for the application. A detailed timing analysis of the CBS compared to other TSN shapers was provided by Zhao et al. (2022). Their results confirm the expectation that CBS introduces delays for the individual traffic classes due to its distributing effect. These delays are the reason for its limited suitability for only slower CD, which precludes its use as primary TSN-MAN-based hardware. However, as Falk et al. (2019) explain, it can still be interesting as an additional downstream functionality in combination with for example SPQ, to save queue memory in subsequent nodes.

Additional possibilities of congestion control for streams include the resource reservation protocols such as MSRP (IEEE 802.1Q, 2022), and RAP (IEEE 802.1Qdd, 2023). To the best of the author's knowledge, their application in connection with congestion control has not been scientifically investigated. The reason might be that the effect of their application does not raise questions. Another congestion control method can be ingress limiting filters which simply cut exceeding traffic at the receiver or edge bridge side (IEEE 802.1Qci, 2016). These too have not yet been examined in connection with congestion control, apparently for the same reasons as the reservation protocols.

All these algorithms work on the data plane of a sender or bridge and reduce egress or ingress traffic peaks or bursts by either stretching them over time or by avoiding or discarding them. However, they have no information about the real load on the network and are therefore open-loop congestion control concepts. Reviewing the open-loop procedures shows that these can only be used preventively but never directly as a reaction to congestion. By contrast, closed-loop congestion control concepts use actual load feedback to control egressing data traffic. Therefore, these promise better solution concepts in the direction of optimal control.

2.4.2 Closed-loop Congestion Control

In contrast to open-loop congestion control, closed-loop congestion control operates with feedback from an actual network load situation. This is measured within the network and returned to the data source. Depending on the control concept, such feedback can range from rather simple mechanisms, such as binary information (Hasegawa et al., 2000), to rather complex facilities that measure the actual current network traffic on the path (Bonomi & Fendick, 2002; Jain, 1998). However, feedback information must always be transported from a certain point in the network to the sender, which involves communication protocol interaction.

Although congestion control is not the focus of this research, closed-loop congestion control methods must be further investigated to determine if they could also be beneficial for load distribution on two or more paths. This is

particularly important since single-path congestion control has the same objective as subsequent per-path flow control of an LDC.

Hasegawa et al. (2000) improved the Transmission Control Protocol TCP (IETF RFC 793, 1981) which is one of the major transport protocols used on the Internet and in campus **Layer 3 based networks**. The original TCP congestion control is a prominent example of closed-loop congestion control and uses a congestion-avoidance algorithm (IETF RFC 5681, 2009). A TCP sender maintains a congestion window value that contains the number of unacknowledged bytes sent out. It uses an additive-increase/multiplicative-decrease (AIMD) scheme to adapt the congestion window size to the network situation. The amount of data sent onto the network path to the receiver slowly increases while receiving positive feedback via packet acknowledgment. Thus, the acknowledgement of the sent packages serves as binary feedback. In the case of a missing acknowledgement the sender immediately cuts the congestion window size (multiplicative decrease) to reduce the load on the network path. Thus, the TCP congestion avoidance algorithm is an end-to-end algorithm as feedback is provided by the receiver end station of the data. However, the algorithm does not consider the stability issues that can occur, particularly with different path delays in various networks.

To provide the possibility of adapting to different network characteristics, Hasegawa et al. (2000), presented TCP Reno, which in principle provides a different and improved increase/decrease strategy. This led then to the TCP NewReno Fast Recovery mechanism officially proposed by IETF RFC 6582 (2012), which provides further strategies for increasing and decreasing the TCP congestion window.

Thereby, different and specially tailored properties regarding the reaction time or selectivity were achieved. However, a major disadvantage, in addition to insufficient closed-loop stability corrections, is that the feedback on congestion, that is, the missing acknowledgement, is generated by actual packet loss which to avoid would have been the innermost goal of any congestion-avoidance algorithm. A further drawback of the on-off-principle is that it is prone to load oscillations.

These disadvantages of the TCP congestion avoidance algorithm, among others, have led to the development of better protocols, such as XCP (Katabi et al., 2002), which facilitates analog feedback for a much better adapted control reaction on imminent congestion. Katabi et al. (2002) showed that TCP becomes inefficient and prone to instability by increasing the per-flow product of the bandwidth and latency. This is becoming increasingly problematic with the application of high-bandwidth optical and large-delay satellite links. They also showed that closed-loop congestion control load oscillations depend directly on the bandwidth-delay product when applying TCP congestion avoidance algorithms. This is the product of the network bandwidth and round-trip time (RTT or rtt), which is the time for data to reach the receiver plus the time the feedback needs to return to the sender. Therefore, Katabi et al. (2002) proposed an eXplicit Control Protocol, XCP, that extends the TCP congestion notification mechanism by introducing a closed loop efficiency controller. XCP achieves better congestion avoidance results than the standard TCP mechanisms.

It is further seen that XCP for congestion control also built the basis for further research and development in the area of load balancing concepts such as TeXCP (Kandula et al., 2005) , where not only the avoidance of congestion is the goal, but also the distribution of load onto other available network paths.

Since then, many other scientists have worked on reducing TCP congestion. For example, Wu et al. (2009) extended XCP to overcome the disadvantage of extensive router calculation burden and introduced a more efficient and fairer explicit congestion control protocol, EFXCP. It operates with longer sample intervals for router throughput and congestion calculations. Further research on closed loop congestion control was provided among others by Bonomi and Fendick (2002) on ATM networks, by Wan et al. (2011) for sensor networks, and by Chen and Khorasani (2011) on large scale networks. Geist and Jaeger (2019) provide a detailed overview of various TCP congestion avoidance algorithms and their characteristics.

The general concept of congestion notification has also been standardized in the IEEE 802.1Q (2022) standard for layer 2 networks.

In recent years, general network protocol congestion control mechanisms for **Layer 3 routed networks** have become the subject of new research, particularly in the field of wireless networks. Kanagarathinam et al. (2020) propose a dynamic TCP (D-TCP) capable of coping with the high channel fluctuations of mobile networks such as mmWave NR and LTE-A. It uses variable estimated bandwidth consumption. Saedi and El-Ocla (2021) introduce "Congestion Control Enhancement for Random Loss plus (CERL+)" which works with an average round trip time (rtt) length. Both D-TCP and CERL+ improve congestion loss in mobile networks with random packet losses. Kasoro et al. (2021) also presented a new algorithm called ABCSS as an optimised combination of the "Appropriate Byte Counting" and "Slow Start" methods, which is based on a variable TCP congestion window size.

Closed-loop congestion control, which focuses on a single path, has paved the way for closed-loop load distribution control, the focus of this study, with the goal of evenly distributing the load across multiple paths.

Summarising this review on general **Layer 3 network** closed-loop congestion control, it shows that numerous improved congestion avoidance algorithms for the TCP protocol have been defined. These achieve better results than the original TCP but all work with traffic-reduction methods. This is fine for general networks layer 3 TCPs lower priority Internet traffic, but inappropriate for CD of automation networks. These are typically not based on TCP but on vendor-specific protocols or dedicated automation transport protocols. Automation CD is furthermore time-critical data and is not allowed to be significantly reduced in throughput. Therefore, these transport protocol-related congestion avoidance algorithms cannot be used directly for automation CD within MANs or TSN MANs. Nonetheless, the important insight gained from the review of the control methods, is that path delays play an important role in the stability of load control. This knowledge can be directly applied on the subsequent flow control for LDC in TSN MANs.

Within smaller **Layer 2** communication subnets or network cells, the transport decisions are not based on IP routing as with Layer 3 networks, but on switching,

based on the MAC-Address of the destination. When evaluating previous research in Layer 2 networks, it was found that this work is often dedicated to either special Layer 2 networks such as ATM networks (Jain, 1998; Kelly et al., 1998; Mascolo, 2000; Yin & Hluchyj, 1994), software defined networks (SDN) (Ma et al., 2017; Prabakaran & Ramar, 2021), sensor networks (Chen & Khorasani, 2011; Wan et al., 2011), or it is reduced to special Layer 1 hardware such as wireless media (Lopez-Perez et al., 2016; Lu et al., 2018; Wang & Hung, 2012) or optical media (Nam-Uk et al., 2009). The review of their applied control methods does not provide new methods compared to those used in the Layer 3 networks. Furthermore, neither have TSN shapers or schedulers been involved in these research projects nor the traditional IEEE 802.1Q defined basic SPQ, which would have been important for TSN MAN. Additionally, they worked with dedicated control algorithms as introduced and investigated in Section 2.3.3. Thus, their control principle is only of secondary importance compared to the best practice linear PID control possibilities.

2.5 Traffic Engineering

The term ‘traffic engineering‘ (TE), in the context of IT networks, was originally used mainly in connection with ISP networks. It defines provisions and control automatisms to optimally use the available multiple intra-domain connections within an Autonomous System (AS) such as university campuses, companies, or ISP domains (Fortz et al., 2002). To achieve this, state-of-the-art Interior Gateway Protocols (IGP), such as Open Shortest Path First (OSPF), Intermediate State - Intermediate State (IS-IS), and Multi-Protocol Label Switching (MPLS) (Goulamghoss & Bassoo, 2020; Jong-Moon, 2000), were enhanced with traffic engineering functionality.

Meanwhile, the relevant network standard IEEE 802.1Q (2022) includes provider backbone bridge TE (PBB-TE) to build a common TE basis for layer 3 and layer 2 networks.

According to Smith (2003) the goals of traffic engineering are :

- To enhance protocols to automatically map packets onto appropriate paths.
- To determine the best paths for data traffic with respect to properties such as:
    - Bandwidth demand and bandwidth availability on the single paths
    - Priority relative to other data priority
    - Delay constraints
    - Media requirements

The crucial property of TE is thus that it typically occurs in the network planning and administration phase and not at network runtime unless new boundary conditions demand corrections at runtime. In some studies, such as the one from Elwalid et al. (2002), the expression ‘adaptive TE ‘ has been used for dynamic load distribution at runtime.

Basically, the literature decides between ISO/OSI Layer 3 and Layer 2 network traffic engineering or the combination of both, then referred to as multiple layer traffic engineering.

2.5.1 Layer 3 Traffic Engineering

As Tanenbaum et al. (2021) describe, most AS internally use traditional IGP such as Routing Information Protocol (RIP), OSPF, IS-IS, or Enhanced Interior Gateway Routing Protocol (EIGRP) on the routers. Basically, routing protocols exchange information with neighbouring routers regarding the quality of the links. This is assessed by assigning a link weight integer value. Each router maintains a complete view of its network domain. The link weight value is derived from the link speed and delay. With these classical routing protocols, the router computes a “shortest” path to the destination by building the sum of the link weights along all available links or paths and by selecting the minimum of these sums. The forwarding decisions are then based on shortest path information.

The literature review has found that there are different concepts for achieving layer 3 traffic engineering using the classical IGP. These approaches can basically be divided into static and dynamic traffic engineering approaches.

Static traffic engineering by favourable administration:

Traditional IGPs such as RIP, OSPF, IS-IS, or EIGRP do not provide mechanisms to support automatic traffic engineering. Fortz and Thorup (2000) show that one way to bypass this disadvantage and nonetheless implement traffic engineering, is to set static routing rules or to adapt dynamic routing rules to a given traffic distribution. This is achieved by manually adapting link weights to a given traffic pattern. This has the advantage that standard routing protocols can be used. Another advantage is that single network links can be adapted according to the global network view of the network administrator, which can achieve a network-wide optimisation of the link loads.

However, the need for constant network re-configurations with each change in the network represents a rather elementary and cumbersome approach. It constantly demands the action of a network administrator or administration tools for automatic reconfiguration following known traffic patterns if these are changing slowly enough. It does not scale with big networks or with fast-changing traffic patterns as they occur in a TSN MAN.

This disadvantage was partly defused by Jong-Moon (2000) by the use of signaling protocols such as constraint-based routing label distribution protocol (CRLDP) and the resource reservation protocol (RSVP). However, it would be still too slow for fast path changing of data traffic as it is the goal for a dynamic LDC in TSN MAN.

Dynamic traffic engineering with enhanced routing protocols:

Notably ISPs have a growing demand to dynamically adapt to changes in traffic patterns and to provide customers with communication connections of a guaranteed Quality of Service (QoS). The disadvantages of slow and elaborate static traffic engineering approaches have led to the development of traffic engineering extensions within traditional routing protocols. For example, MPLS TE (Smith, 2003), also known as IP/MPLS TE, or OSPF-TE (IETF RFC 3630, 2003).

Tanenbaum et al. (2021) explain that MPLS is an integration of layer 2 technology into layer 3 technology, as it assigns labels to data packages. The routers channelize the data packages via these labels onto certain network paths without considering layer 3 content which is located further inside the data frames. MPLS defines one or more tunnels, the so-called labeled switched paths (LSP), from the ingress point (LSP head) to the egress point (LSP tail). If the data have crossed the tunnel and have reached the egress point of the network, usually at the other end of the provider backbone, the label information is removed again. The MPLS itself can only provide different channels to apply load distribution but has no means for distribution decisions. MPLS TE assigns enhanced metric information to LSP tunnels and performs traffic engineering calculations to assign traffic to the tunnels to meet the distribution requirements (Goulamghoss & Bassoo, 2020; Smith, 2003). It uses traditional IGP extensions on OSPF or IS-IS to distribute enhanced metric information to the head-end calculation modules. As Goulamghoss and Bassoo (2020) explain, a further possibility to enhance the rerouting convergence speed after a change in the network is to apply MPLS Fast Reroute (FRR). Thereby, backup paths are pre-computed and pre-established along the LSP before link or node failure issues. This can reduce convergence time to about 100 ms which would be even fast enough for slower automation applications CD. However, MPLS is typically a network function of higher layer networks and is typically not supported by MAN devices installed at the field- and controller-level networks. Therefore, MPLS TE cannot be considered for a MAN TSN LDC solution that are the focus of this research. It can only be an alternative for higher-level MANs running Deterministic Networking (DetNet) according to IETF RFC 8655 (2019). DetNet is a deterministic approach for Layer 3 networks, similar to TSN for Layer 2, which can be based on either IP or MPLS.

2.5.2 Layer 2 Traffic Engineering

In addition to the application of Layer 2 technology to reach a higher level of automatic traffic channeling in Layer 3 networks, as described in the previous sub-section with MPLS TE, Layer 2 traffic engineering, as investigated for example by

de Sousa and Soares (2008), mainly implies the static application of the traffic control means provided by the classical Layer 2 methods defined by IEEE 802.1Q (2022). This includes primarily the logical separation of the network by Virtual LANs (VLAN) assignment and the Quality of Service (QoS) assignment to certain network traffic. In combination with the multiple spanning tree protocol (MSTP) (IEEE 802.1Q, 2022), the assignment of favourable path costs is an additional means to influence the selection of data paths.

The assignment of VLAN Identifiers to the physical ports of bridges of a given network, as described by Tanenbaum et al. (2021), creates logically separated LANs within a single physical LAN. MSTP allows the operation of these virtual LANs as separated network spanning trees. These trees can be designed to be derived from different root nodes and thus form Multiple Spanning Tree Instances (MSTI). With MSTP TE, solutions such as those applied by de Sousa and Soares (2008) and Santos et al. (2009), use the MSTP path cost and root selection to arrange this set of MSTI in a favourable way to minimise the maximum network loads on the overall paths. However, the MSTP reconfiguration time typically takes a few seconds, which is too slow for MAN applications. Additionally, configuring an MSTP region with different VLANs and MSTI is a complex process which demands IT specialist knowledge that is not typically present in the MAN application domain.

The MSTP TE has not been used for better load sharing only. Ali et al. (2005) propose the use of different MSTP regions to divide the network domain into smaller parts. Thereby, they achieve better results for network convergence time as a reaction to network changes, reduced influence of network failures, and better failure localisation in addition to a better network utilisation. Nevertheless, de Sousa and Soares (2007, 2008) showed that the single MSTP region approach is most effective in terms of both load sharing optimisation and service disruption. Santos et al. (2009) built on this knowledge by optimising link load balancing in single-region MSTP networks. They investigated two optimisation objectives. First, the minimisation of n worst link loads, with n up to the total number of network links, and second, the minimisation of the average link load, when n is

less than the total number of network links. They solved this task by appropriate modelling of the network and the use of standard linear programming optimisation and showed that the solution of models of only a reasonable network size already create excessive computational effort. Therefore, they proposed the application of heuristics based on the optimisation of a relaxed problem, where only fractional network parameter assignments to the spanning trees are allowed, from which feasible solutions can be derived. However, the results for the reconfigurations are still in the range of a few 100 ms, despite the very high computational effort for the optimisation.

To find the optimally shared load, Ho et al. (2011) used the "Local Search" (LS) method to find a global minimum maximum of the network utilization. Local search is also a heuristic method for solving complex optimisation problems. They used the COMET (Comet, 2023) optimisation tool, which is a dedicated programming language used to solve complex combinatorial optimisation problems, to simulate the MSTP network. The simulated network consisted of a set of N switches (nodes) and a set of E links (arcs or edges). Assigning k MSTIs to this network yields the following:

1. k initial link cost matrices $\boldsymbol{W_1}$, $\boldsymbol{W_s}$, … $\boldsymbol{W_k}$,
2. a bandwidth matrix $\boldsymbol{BW}$
3. k traffic demand matrices $\boldsymbol{TD_1}$, $\boldsymbol{TD_2}$,… $\boldsymbol{TD_k}$,

The objective of the optimisation was to find *k* MSTIs for the *k* VLANs to minimise the maximum of the link utilisation U_{max} of all possible links *e* within *E*. Thereby, in this approach, the search space is made of the spanning trees, not of the link costs as in previous approaches. However, constantly changing traffic demand matrices over time because of unpredictable network user behavior create constantly challenging high effort optimisation computations of a central instance and a continuous collection of all network information. The latter also causes considerable additional traffic.

These MSTP solutions all achieve the goal of a well-balanced network load. However, they are only suitable for relatively slowly changing traffic patterns. An

MSTP reconfiguration typically takes a few seconds (IEEE 802.1Q, 2022), which is slow for the goal of quick reactions to load changes in a MAN with communication cycles of a few milliseconds. Another key disadvantage of an MSTP solution is the risk that carelessly applied topology changes can result in the reconfiguration of the entire network tree, which would result in unacceptable application downtime.

Another form of Layer 2 traffic engineering, which was among others investigated for example by Wang et al. (2021), is the establishment of dedicated send slots for talkers participating in TSN networks. They achieve resource assignment using, for example, the EST provisions defined by IEEE 802.1Q (2018). In this case, the traffic engineering is performed using CNC that optimises the send slots of synchronised talkers and bridges. Several algorithms have been proposed for this purpose. For example, Wang et al. (2021) applied an improved ACO called IACO, to schedule time-triggered stream transmissions. They define minimal send slots of frame length and demand that only one node is using a certain send slot. Thus, they achieve an optimum load distribution. However, the necessary complete recalcualations after a change in the network make it an inflexible approach, not suited for MAN.

Li et al. (2022) used a joint routing and scheduling algorithm that achieved recalculation for a few thousand streams at the sub-second level. The central recalculation of the complete time-critical traffic, however, also has the major disadvantage of inflexibility regarding added or removed network participants or links. This typically again demands a complete schedule reconfiguration, and thus a reconfiguration of all network participants which disturbs the automation applications.

For Ethernet networks according to IEEE 802.1Q, as they are the basis for this thesis, facilities for TE are thus far only specified by standards purely for use within Provider Backbone Bridged Networks (PBBN). These are based on the definitions of IETF RFC 5305 (2008), IETF IS-IS Extensions for Traffic Engineering, and IS-IS Traffic Engineering Metric Extensions of IETF RFC 8570 (2019). These PBB-TE mechanisms use path control and reservation (ISIS-PCR) to find multiple

favourable paths and set up a Traffic Engineering Database (TED). This is propagated by IS-IS, which stores the traffic engineering information in each PBB. Crucially, however, is that bridges suitable for industry and MAN typically do not support ISIS-PCR. Thus, this means of pathfinding is not available for this research. Furthermore, the actual use of multiple paths, that is, any load sharing or load distribution concepts or algorithms, as shown in the previous sections, are not part of the IEEE 802.1Q. These are left to be designed by the user of the network, using controller types as listed in Section 2.3.3.

2.5.3 Multilayer Traffic Engineering

When network operators migrated networks to optical IP-over-WDM (Wavelength Division Multiplexing), so called multilayer networks, it was necessary to define mechanisms that allowed to make use of the resources offered by both layers in a coordinated manner. This led to the definition of Generalized Multi-Protocol Label Switching (GMPLS), as defined by IETF RFC 6002 (2010), on the control plane, which allows automatic set up and tearing down of light paths in the data plane.

As for example the research of Puype et al. (2009) shows, Multilayer traffic engineering (MLTE) provides cross-layer network optimisation techniques to cope with short-term evolution or rapid changes in traffic patterns. It extends the Layer 3 IP/MPLS TE towards Layer 2 MPLS-over-optical network traffic engineering by integrating Layer 2 optical switching optimisations. The multilayer approach features a much higher flexibility to network changes compared to a single layer TE solution and is therefore especially suited to serve in multiservice environments. In contrast to other approaches, that propose a reactive approach in the case of network overload, such as the topology reconfiguration mechanism from Gillani et al. (2005), Puype et al. (2009) work with a more proactive approach. Their solution is based on three integrated mechanisms that continuously optimise network performance by analysing traffic measurement data, that is, this TE can be classified as a dynamic and closed-loop approach. The use of different mechanisms is necessary due to the different needs of a variety of data from

different service classes. For example, bulky BE data is managed by using rather slow and inflexible layer 3 mechanisms, whereas selected services requiring higher QoS such as CD are assigned to faster acting Layer 2 mechanisms. The latter, which are also the focus of this study, work with dynamically modified link capacities, independent of actual logical topology connectivity. For this, an optical light-path setup and teardown is required but the logical topology is not changed. The maximum admissible link capacity is crucial for the network formation. However, neither wired nor optical MAN and TSN MAN paths provide the feature of setting up or tearing down additional light paths. Thus, this approach is not available for TSN MAN.

Another promising MLTE approach was provided by Lopez et al. (2010), who defined an algorithm that efficiently manages the resources from both layers equally. Their concept is based on the Bayesian Decision Theory, that is, they use load statistics for the path selection decision to find a compromise in assigning the optimal number of label-switched paths that are to be switched over the electronic and optical network domain sections. The result is a heuristic dedicated algorithm. Also here, the method of analysing the load to adapt load-dependent link costs for path selection is in principle an interesting approach also for the MAN LDC goals of this thesis. However, again, there is no possibility in a wire-based or optical MAN to set up additional paths or bandwidth capacity in a load-dependent manner, as with the IP/MPLS forwarding adjacencies for light paths or capacity up/-downgrading.

2.6 Load Balancing

The concept of load balancing has long been known in client/server systems (Cardellini et al., 1999), distributed systems (Zaki et al., 1996), and network operations (Elwalid et al., 2002). As the literature review shows, the expressions “load sharing” or “load distribution” are often used as synonyms for “load balancing,” while the latter expression clearly dominates. To be precise, “balancing” in its original meaning rather stands for an equal distribution whereas

the expressions “distribution” or “sharing” do not necessarily demand equal partitions. In this thesis, all three expressions are used as synonyms.

The main areas where load sharing or load distribution concepts are applied are typically server load balancing (SLB), distributed systems load balancing (DSLB), cloud computing load balancing (CCLB), and network load balancing (NLB). Although only network load balancing is important for this thesis, it is important for reasons of delimitation to review all four.

2.6.1 Server Load Balancing

For many IT services such as e-mail, storage or web applications, client/server systems are set up (Tanenbaum et al., 2021). To provide scalable system capacity and performance in combination with higher availability, the setup of multiple servers forming a server cluster is common, as shown in Figure 2.9.

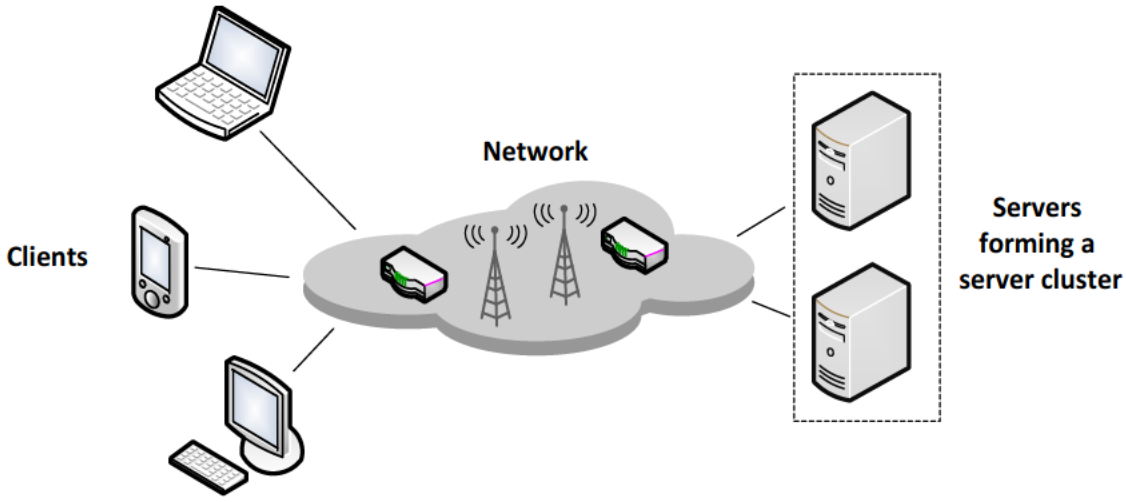


Figure 2.9: Client/Server system with server cluster

In addition, the challenge of achieving an evenly distributed server load within the server cluster to avoid a possible overload of single-server hosts arises. This requirement led to the emergence of SLB concepts.

Summarising the statements of Cardellini et al. (1999), the goal of SLB is to continuously distribute a variety of client requests onto several available

servers of the server cluster to improve the available services with regard to:

- Scalability
- Availability
- Load balancing
- Applicability to a variety of services
- System updates without down times

Thus, the SLB is a distribution of load among the end stations, regardless of its path loads. Path loads, as they are important for this research, are not primarily the focus of SLB. However, to a certain extent the load on the paths to the servers is also balanced if they are accessed through different paths.

The number of servers in a cluster is usually allowed to vary in numbers. Traditional early SLB applications, such as those described by Cardellini et al. (1999), were laid out for a smaller number of typically two or three servers.

They describe various client-based and server-based packet routing strategies, all of which can be classified as oblivious routing algorithms as described in Section 1.1. They are thus not closed-loop algorithms that could help bringing this research to an optimal dynamic LDC for TSN MAN.

Subsequent research, such as that provided by Wilson and Deepalakshmi (2019), targeted higher numbers of servers or clusters. In addition, they proposed a dynamic server load balancing algorithm (DServ-LB) using SDN-switches for dynamically varying the number of servers. The proposed control algorithm routes requests to the least loaded server using load parameters such as remaining memory, remaining CPU load, and number of available connections. These three parameters are combined by a central controller according to an empirically obtained calculation rule. The controller algorithm is thus to be classified as a dedicated closed-loop control algorithm according to Subsection

2.3.3. However, such an algorithm cannot be directly reused for a network LDC that depends solely on comparing the link data load. A proven PID controller for the link load might be the better solution here. The use of SDN switches is only important in terms of how the control decision is brought to the data plane to multiplex requests to the servers. However, this is not important for network LDC in MANs when distributed LDC-Controllers are part of each AC and decide their own send-paths for their individual own traffic on their own ports.

A similar approach, also within SDN, was provided by Bojović and Živko (2022), who also used a multi-parameter closed-loop dedicated control algorithm to measure actual host loads. The same restrictions apply here regarding a re-use for TSN MAN as with the Wilson and Depalakshmi approach.

Other well-known implementations of contemporary SLB for a higher number of servers are the Microsoft Network Load Balancing (Microsoft, 2023), where the expression “Network” is used as a synonym for servers, and the Linux Load Balancer is a part of the Linux Virtual Server (LVS) (Linux, 2023) software. Here, too, dedicated algorithms were used that work with server parameters and cannot be directly transferred to a MAN network LDC. In conclusion, however, it must be stated for SLB in general that the transferred idea of distributing the load to different MAN devices in a ring, in this case ACs instead of servers, does not necessarily lead to distribution over the paths, which is the goal. However, it would be expected that the spatial separation of several ACs in a ring would also result in better load distribution in the ring, since data is sent to and from the ACs in different directions. For controller types, however, no new controller approaches could be identified in the SLB literature apart from those listed in in Section 2.3.3.

2.6.2 Distributed Systems Load Balancing

In addition to the trend of clustering internet servers, the growing availability of low-cost high-performance CPUs in connection with highly effective

communication networks has enabled the distribution of extensive calculation tasks on a variety of hosts. Also, within such distributed systems, the challenge is to balance the task loads. Both static and dynamic load balancing solutions were applied, as described by Taley and Keole (2015). These distributed systems load balancing (DSLB), or load distribution concepts also clearly target the relief of connected systems rather than the relief of the connecting network, which is the focus of this research. Nevertheless, the applied balancing concepts also contain interesting aspects for network de-loading goals. Examples of early concepts for distributed systems started with the load distribution for a network of workstations (Zaki et al., 1996) which were loaded by parts of a calculation task proportional to their individual performance. The driving goal of this was, and still is, to parallelise sub-tasks to minimise the execution time of the complete task. In addition to previous static approaches, where tasks and resources were limited to certain borders, Zaki et al. investigated different dynamic load distribution schemes with varying program and system parameters. Static load balancing for distributed systems is performed at compile-time, whereas dynamic load balancing (DLB) is executed at runtime according to the changing number of users and their changing applications. According to Zaki et al., four steps are necessary for DSLB:

1. Monitor processor performance.
2. Calculate new distributions.
3. Make work-distribution decisions.
4. Move the data to the distributed systems.

They chose four different schemes to construct the load sharing mechanisms:

1. The concept is based on either local, or
2. global information to base the load balancing decision on it, and it is,
3. either centralized, or
4. distributed, depending on whether the load balancer is situated in one system or distributed among the systems.

These four poles, local versus global and centralized versus distributed, are also important in the design of load sharing for general networks and MAN, as discussed in Chapter 4 .

Metawei et al. (2012) presented another approach: an agent-based DSLB. They used a credit-based system in which each agent in a system is assigned credit that decides whether it will be migrated or transferred to another system. Multiple linear regression, calculated using a super-ordinated system, was applied to reach the migration decision, based on a multitude of regression parameters such as computational load, resource availabilities, and communication reliability. It is thus very similar to the parameter-based SLB approach of Wilson and Deepalakshmi described in Subsection 2.6.1. However, like the Wilson and Deepalakshmi approach, it cannot directly be used for TSN MAN LDC either. Furthermore, the idea of using a credit system to decide heavily or lightly loaded systems is similar to a port load parameter at each port within a MAN LDC to decide heavily or lightly loaded ports. This approach therefore does not reveal any new ideas for an optimal MAN LDC solution. Later research focusing on higher-number distributed computer systems was mostly published in areas of massively distributed systems or cloud-computing systems, as described in the next section.

Looking at DSLB, it must be noted that the objectives and methods used are very similar to those of the SLB. The focus is more on relieving the systems than on relieving the transportation network. For this reason, the methods used would only be useful for NLB in TSN MAN to the extent that physical separation with dedicated access paths could be assumed. However, this is not the case for the underlying TSN MAN rings. Nevertheless, the methods used for bandwidth capacity utilization measurements and control are also of interest for TSN MAN NLB. However, no other than those already listed in Section 2.3 are used.

2.6.3 Cloud Computing Load Balancing

The further development of SLB and DSLB was to extend the originally limited number of servers in server clusters to a significantly higher number of servers, including the possibility of geographically distributed systems. This has led to the development of cloud computing load balancing (CCLB) systems, providing the advantage of virtualization and geo-redundancy. An overview of numerous CCLB algorithms was provided by Rajeshkannan and Aramudhan (2016). Rajeshkannan et al. classified CCLB algorithms as software- and hardware-based approaches. The former are running on communicating machines in the cloud itself. The latter are located in front of a cluster and route all the traffic among the servers. Examples were obtained from the simple round-robin algorithm to the rather complex ACO, but no further controller types than those listed in Subsection 2.3.3 are visible in their review.

A more recent systematic literature review analysis of the existing dynamic CCLB was conducted by Tawfeeg et al. (2022). They classified dynamic algorithms into

1. sender initiated, when the sender starts the load balancing process,
2. receiver initiated, when the receiver is responsible for the load balancing process, and
3. hybrid, when the algorithm is a mixture of 1. and 2.

They found and analysed more than 40 different algorithms presented by the research community since 2015. These are essentially also based on the application of the methods presented in Section 2.3. The principles of the solutions are comparable to those of SLB and DSLB. In addition, various specialised algorithms have been designed. One interesting example that also included edge stations, that is, stations that are located between a cloud and a physical automation application and are thus interesting for MAN, was provided by Nezami et al. (2021). They propose a solution, which introduces a decentralized multi-agent system for collective learning that utilizes edge-to-cloud nodes to jointly balance the input workload across the

network. However, this cannot be mapped to a TSN MAN with their centralized ACs in a field level or controller level ring.

Also, with CCLB, as with SLB and DSLB, the methods used for capacity utilization measurements and control are similar to those stated in Section 2.3 and as such are also of interest for TSN MAN network LDC.

2.6.4 Network Load Balancing

As Tanenbaum et al. (2021) explain, Network Load Balancing (NLB) or network LDC (NLDC) aims to avoid load peaks on individual network paths, that is, it distributes traffic across all paths. However, sometimes the expression NLB is also used as a synonym for "Server Load Balancing" (Microsoft, 2023) (see Section 2.6.1). This is because servers might be reachable over different network sections. Distributing loads on different servers can thus also imply the distribution of loads onto different network paths. In its original meaning though, NLB is used to distribute network traffic among several network paths.

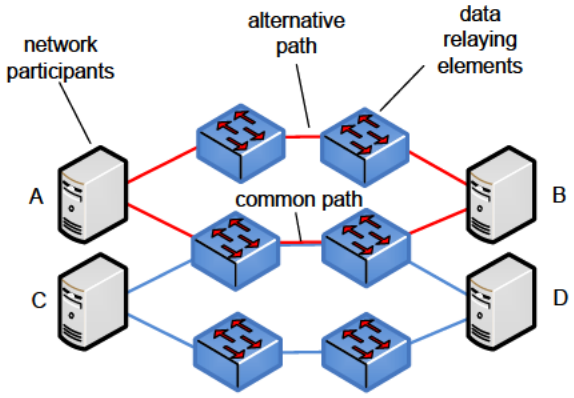


Figure 2.10: Common path in redundant or multi-paths communication networks

Setting up the communication network as a multi-paths network, as shown in Figure 2.10, offers alternative paths (Tanenbaum et al., 2021) that can be used for load distribution. Depending on where the multiple paths are rooted, the NLB can be located at the end stations or in the bridge or router.

In **OSI Layer 3 networks,** NLB is used to distribute network traffic from one router to the next router using alternate routing paths (Cisco, 2022; Tanenbaum et al., 2021). Antic et al. (2010) analysed the possibilities of NLB in connection with standard Shortest Path Routing (SPR) protocols, for example the standard routing protocol OSPF. They presented a Load Balancing Shortest Path Routing (LB-SPR) method which achieves a balancing of arbitrary traffic patterns by using a two-phase routing scheme. In the first phase, portions of traffic are routed to all possible intermediate routers towards the destination, including on the shortest path. In the second phase all intermediate routers forward to the destination. Thus, they achieve a traffic distribution. However, this method does not consider interfering exogenous traffic from other sources which might load links along the path already. It can be regarded as an oblivious method therefore according to Wang et al. (2006) (see Subsection 2.3.1), without actual closed-loop feedback which could be expected to optimise the load distributions.

For protocol-based NLB, Antic et al. and also Fortz and Thorup (2000) extended these shortest path routing protocols, such as OSPF, by path weight metrics, so-called costs, to support routing decisions. Load balancing can either operate on equal cost or equal distance paths or on unequal cost or unequal distance paths. When working with the classical pure routing protocols RIP or OSPF, load balancing can only occur if the router has installed multiple paths that are equivalent in administrative distance and cost. When applying IGRP or EIGRP, unequal paths can also be subject to load balancing. However, this principle of using costs as routing decision criterion is not well suited for MAN rings where the distance from talker to listener is typically different for the two directions of the ring.

A similar procedure was provided by Chadha and Gupta (2013), which apply equal cost LB and unequal cost LB. With the former, they apply LB if the paths have equal costs. With the latter, they only use the lower loaded path if the costs are unequal. They gain the costs themselves from the path delays. It must be objected though, that path delay can only serve as a decision criterion for paths of equal costs. Then,

typically the load is causing the additional delay. For MAN rings the delay is therefore not significant as it is different in most cases in the two ring directions.

The LDC for the NLB can also be independent from the routing protocol. It is then located in a higher application layer. Elwalid et al. (2002) proposed several possibilities for the actual distribution process, independent of the use of a specific routing protocol:

1. On a per-packet basis: each new packet is routed over the next path interface being part of the load balancing paths.
2. On a per-flow basis. Flows are frames that have a <source IP address, source port, destination IP address, destination port, and IP protocol>-tuple in common.
3. On a per-destination basis: each new destination flow is routed over the next path interface.

As emerges from their work, the destination-based method has the advantage of preserving the packet order but has the potential disadvantage of unequal usage of links. This method is typically applied to a higher number of destinations. The per-flow approach has the disadvantage of a rather high traffic analysis and state bookkeeping effort but also preserves the packet order. The packet-based approach bears the risk of packets arriving out of order at the same destination when they are delivered over different paths. Transferring this to the MAN rings and CD traffic, none of these seems an ideal method. The per-packet selection of different paths bears especially in rings the out of order arrival risk. MAN-CD are typically no IP or TCP data, that is, the flow-based method is not applicable either. The per-destination method does not take the fact into account that different destinations can have different CD traffic demands. Therefore, from MAN point of view, the Elwalid et al. list of methods should be expanded by the per-automation-application flow of CD.

Kandula et al. (2007) presented FLARE (Flowlet Aware Routing Engine), a method that overcomes the packet order problem by keeping the inter-packet gap higher than the longest path delay. However, this could have a negative influence on the

CD data flow particularly in bigger MAN rings with long path delays. The per-flow-based approach could also be used if different flow-assignment criteria would be used. MAC-Addresses instead of IP-Addresses for example.

Dynamic network load control, as investigated by research in layer 3 ISP networks or campus networks (Ahmad & Khan, 2018; Elwalid et al., 2002; Lopez-Perez et al., 2016; Neely et al., 2008; Wang et al., 2006) and introduced in Section 1.1, usually differentiates between flow control, fairness control, and distribution control. As also outlined in Section 1.1, fairness control is not relevant for automation network control data, as CD data flows are not allowed to be reduced, as explained further down. Regarding distribution and flow control, more research has been conducted on distribution control than on the flow control subtask. This results from reviewing the research on closed-loop network load control as listed in Subsection 2.3.2.

In **OSI Layer 2 networks**, the bridging standard (IEEE 802.1Q, 2022) also considers the increased need for load sharing. Therefore, it defines the facility of Enhanced Transmission Selection (ETS), which allows the network user to assign priority-based processing and bandwidth allocations for different traffic classes. This mainly aims at Data Center Bridging (DCB) networks, which are rather IT layer 2 networks, but could also be used in layer 2 networks for automation. Examples of actual methods or necessary algorithms for the load-sharing function are not defined in the standard, which encourages continuous research, improvement, and vendor delimitation. A survey on NLB in the area of DCB Networks was conducted by Zhang et al. (2018). They compared several DCB-dedicated load balancing algorithms such as Freeway (Wei et al., 2014), Fastpass (Perry et al., 2014), or “flow distribution aware load balancing for data centre networks (FDALB)” (Shuo et al., 2016), to name only a few of them.

Wei et al. distinguish between so-called long-lived elephant flows and latency-sensitive mice flows. They leverage on the presence of multiple shortest paths and use a scheduling scheme to adaptively partition the transmission paths into low latency paths and high throughput paths for the mice and elephant flows

respectively. They propose a dedicated algorithm (Freeway) to dynamically adjust the number of separated transmission paths.

Perry et al. transfer the path selection for traffic transmission from the data centre endpoints and routers to a central arbiter. This Fastpass network architecture provides two dedicated algorithms to assign transmission time and path selection to the endpoints and routers. They claim that they achieve a considerable reduction of the queuing resources by maintaining a high throughput and low latency.

Also, Shuo et al. propose a central solution, FDALB, to reduce flow collisions and achieve a high scalability. Similar to Wei et al. they classify short flows and long flows but only centrally manage the long flows, whereas the short flows are managed by the distributed switches themselves. In addition, end-hosts shall tag long flows for the switches to easily determine the long flows by inspecting the tag.

These three approaches can all be classified as “dedicated algorithm control,” as described in Section 2.3.3, but cannot be directly re-used in TSN MAN environments for CD. The reason for this is that these algorithms use multiple DCB paths between any source and destination, some of which are reserved for bulky BE traffic and some for shorter, delay-sensitive control data. However, MAN ring topologies only provide two possible paths, which must be available between all nodes in the ring for all types of traffic to ensure accessibility.

In the context of TSN, Nayak (2018) studied the scheduling and routing of time-triggered traffic. He worked with pre-calculated routing and scheduling algorithms. However, for a set of one thousand data streams, these calculations still take several hours which is considered too long if dynamic traffic changes or application launches are to be possible as assumed in the MAN of this research. In addition, he based his work on SDN bridges. However, SDN-based TSN MAN are the exceptions and are not the focus here, as they require dedicated SDN-capable bridges which are often unavailable for industrial bridges for MANs and TSN MANs in particular. Another disadvantage of SDN is that it implies that routing is managed by a

CNC and not in a distributed fashion as is intended to be the basis of this research.

Other recent studies on NLB in combination with ML and SDN were conducted by Todorov et al. (2020) and Jahde et al. (2021). The latter use a Deep Learning (DL) ML approach. As explained in Section 2.3.3, ML is basically suitable for TSN MAN NLB, but it is not the first choice due to the high implementation effort, memory, and CPU resource consumption. Further contemporary research was provided by Han et al. (2021) who adapted the network topology to a traffic forecast via AI. However, this is only suited for inert changing traffic patterns, which is not the case for CD within MANs. Prabakaran and Ramar (2021) also conducted research on NLB with SDN and use the SDN for a rather untypical distributed control concept. Regarding this procedure, it must again be objected that, because it is also an SDN solution, it is of secondary importance in this research.

2.7 Chapter Summary

The overall aim of this thesis is to research, design, develop, and validate a method for optimum control of dynamic load distribution in time-sensitive communication networks for manufacturing automation. As such, this chapter has first reviewed definitions and provided an overview of general communication networks, time-sensitive communication networks in the manufacturing automation domain, and the control theory applicable for communication traffic load reduction or distribution, before considering existing methods for congestion control, traffic engineering, and load balancing applicable for optimum dynamic load control in a TSN MAN.

It has shown that no dedicated research on the problem of load distribution in TSN MANs is available.

In particular, the following gaps exist in the current relevant knowledge to solve this task:

- The influence of various TSN traffic shapers and schedulers on data flow

control must be analysed.

- The implications and connections among automation application cycles, packet sizes, communication cycles and network topology extensions must be investigated.
- The influence of stream reservation, frame preemption and media redundancy protocols must be assessed.
- The best-suited controller types for data flow and distribution control for a TSN MAN with their own properties must be selected.
- It is unclear whether a decentral or a central control concept is to be preferred.
- A dedicated distribution control approach for a single AC featured TSN MAN field-level ring must be proposed. Furthermore, this must be extended for controller level rings featuring multiple ACs.

To solve these problems, it is important to propose, design, develop and validate a new method for optimum control of dynamic load distribution in time-sensitive communication networks for manufacturing automation.

Chapter 3 Research Methodology and Design

This chapter presents how the research project aims to find an optimum control of the dynamic load distribution in communication networks for manufacturing automation. It describes the applied research methods and methodology. Furthermore, it outlines the methods for data collection, data analysis, and how the results are presented.

3.1 Philosophy and Methodology

The research paradigm that underlies this research endeavour is positivism, as it is the most expedient research paradigm to provide highly diagnostic quantitative data to answer the research questions embedded in control theory and communication network theory as part of the natural sciences. This position is supported by for example Crotty (1998) and Grix (2019).

To answer the research questions defined in Section 1.4, a combined research method approach consisting of two methods shall be applied.

Firstly, literature review is used to identify relevant automation communication use cases (IEC/IEEE 60802, 2018), relevant control theory concepts (Duriez et al., 2017; Goodwin et al., 2001; Müller & Guido, 2017; Normey-Rico & Camacho, 2007), and relevant network standards (IEEE 802.1Q, 2022; IEEE 802.1Q TSN TG, 2022).

Secondly, to observe the network behaviour for data gathering, basically two possibilities would be at hand:

1. to carry out experiments with real network bridges and devices, or
2. to simulate the network with the help of simulation tools.

Simulation is the preferred and selected research method to obtain the primary research data to be analysed within this research project. As Wehrle et al. (2010) outline and many applications show (Henderson & Imputato, 2023), network simulation has several advantages compared to experiments:

- Network simulation allows a relatively effortless change of network parameters, such as the number and types of bridges and end stations, traffic types, or topology changes. Changing all these network conditions in a real physical communication network requires a multitude of effort and time compared to the simulation method. The simulation environment provides diagnostic access to all protocol layers of virtual network devices.
- A detailed and simultaneous view into the network device's behaviour and network communication data is possible and delivers more precise and detailed data than real hardware. This is particularly true with respect to the possibility of complete network data snapshots that would require precise time synchronisation in real hardware.
- The network devices and the network can be analyzed in "slow-motion."
- A data logging and data analysis environment is part of the simulation tools, saving the application of further network logging and analysis tools.
- The quality of the simulation data compared to the experimental data is more detailed and richer, as any network detail can be displayed and related to other events at nanosecond resolution. Hence a network simulation environment is an excellent tool for analysing complex relationships with reasonable effort.

A disadvantage of the simulation method is that it lacks the device complexity of real hardware setups. Real bridges and end-stations usually run a variety of other software, such as applications or protocol stacks, in parallel to the functionality in the focus of the research problem. Therefore, the validation of hardware sometimes reveals interaction problems that can be grounded both in design deficiencies and in performance or resource problems of the target hardware. However, because the research problem concentrates on the basic mechanisms of load control and controller interactions, the performance or resource problems of dedicated hardware or software implementations are of secondary importance. Interaction with other protocols or applications on possible host devices of load controllers is not the focus either. Therefore, these simulation disadvantages do not become important to answer the research questions.

A further reason that excludes real hardware experiments is that, at the time of writing this thesis, it was not possible to obtain TSN bridges with the support of the necessary traffic shapers and schedulers and other features such as bandwidth resource reservation or frame preemption. Even if this were possible, the next problem would have been to adapt the bridges with firmware extensions to support the necessary sophisticated bandwidth measurements and feedback mechanisms. A further problem would have been to adapt automation controllers to provide synchronised transmission of single frames. These tasks would have exceeded the possibilities of this study by far.

Figure 3.1 illustrates the complete research methodology.

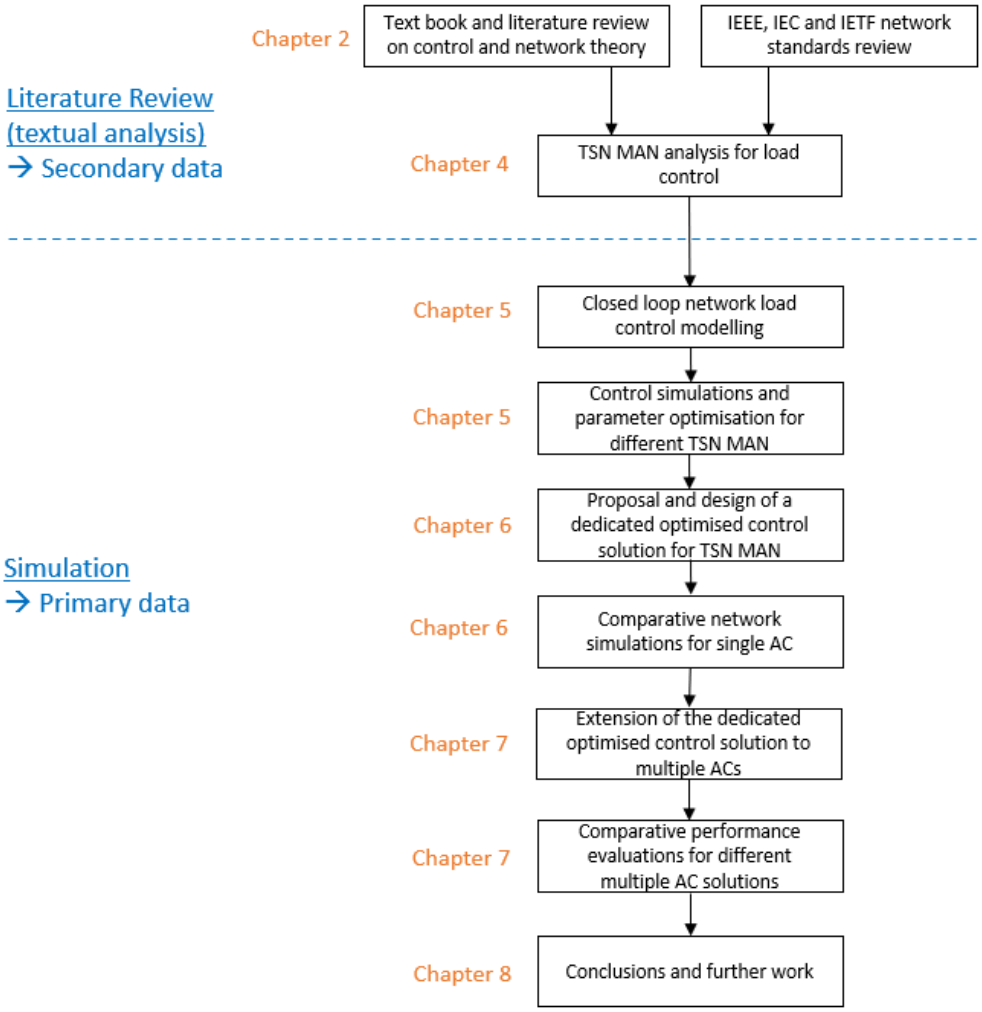


Figure 3.1: Research methodology

Thus, combined research method is applied into this research project. The first

step is a literature review which provides the secondary data. The second step is simulation to provide the primary data.

The literature review part begins with three main tasks:

1. Control theory and network theory concepts must be evaluated regarding their properties and suitability to help investigate and solve the research problems.
2. Relevant IEEE, IEC, and IETF communication network standards must be identified and reviewed, and areas of influence on the research questions must be studied, analysed, and related to the research problems.
3. Analysis of the properties of relevant types of TSN MAN on which research should be focused. This is carried out by studying the relevant automation network applications to identify the application areas of interest and to delimit them to others.

These tasks are prepared with a careful literature review in Chapter 2 and lead to the analysis of TSN MANs in Chapter 4 .

The data gathered during the literature review phase are the input to the second phase, network modelling and simulation. Chapter 5 presents the design and simulation of the control circuit for different TSN traffic shapers and schedulers. Using the literature review results, a model of the network itself, without any load control mechanisms, is designed. This network communication model forms the control “plant” of the closed loop control system. It consists of the mathematical model of one relevant TSN bridge, extended to the model of several bridges connected in series to a line-topology communication network. The mathematical model of the line topology is then simulated using the mathematical simulation environment MATLAB. This tool provides special extension modules to simulate control systems or control engineering problems, namely, Simulink. Knowing the behavior of the communication network as a dynamic system, a proper control algorithm is designed and tested in the closed loop dynamic model using MATLAB/Simulink. With the help of the simulation tool, it is then iteratively

improved. The results and recommendations for the use of different traffic shapers and schedulers in connection with LDC are derived.

In Chapter 6 , based on the findings of Chapter 2 , Chapter 4 , and Chapter 5 , a new dedicated control method optimised for the use within TSN MANs is proposed. With the optimised model of the closed-loop control system of the automation communication network, the network bridges, end-stations, and closed-loop controller are then implemented in software using the network simulation tool ns-3 libraries. Ns-3 is an open-source network simulation environment that is widely used in the communication network research community (ns-3, 2023; Wehrle et al., 2010). These software modules are then integrated into an ns-3 network simulation environment. Complete networks including communication devices such as automation controllers and end stations are simulated. The devices communicate over the network bridges while the load is controlled by closed-loop controller instances residing on the ACs.

In Chapter 7, the controller design is further optimised for the application in networks with multiple AC. Some TSN traffic shapers and schedulers imply mutual controller instances interdependencies, whereas others do not. For both possibilities a solution is proposed. Performance evaluations are provided for both solutions.

Chapter 8 concludes the research project with main achievements, the contributions to the new knowledge generations, discusses the limitations, and proposes further research steps for future projects.

3.2 Methods for Data Collection

The data to be collected

Two types of data are collected to achieve the research objectives: secondary and primary. The secondary data are those for achieving all four research objectives though mainly for Research Objective (RO) 1. The primary data are those for achieving ROs 2 to 4. The secondary data are obtained from the Literature Review (LR). The primary data is generated through simulations.

Method for Secondary Data Collection

Secondary data are collected by analysing previous research work in the area of load control in ISP networks, campus networks, mobile access networks, and cloud computing networks. The various concepts of closed-loop network load control solutions are identified, classified, and evaluated in terms of their capabilities to serve as a basis for further development to solve the specific load control challenges of TSN MAN. For this purpose, the collected secondary data will serve as the background to compare the properties of the new control concepts. Furthermore, the IEEE, IEC and IETF relevant network standards for automation networks are analysed and categorized regarding the types and properties of traffic schedulers, traffic shapers, and data transport protocols. Textbooks on control theory and network theory provide a theoretical basis for the development of the control concept.

Method for Primary Data Collection

The primary data is collected by simulating the manufacturing automation network control model and by simulating typical complete network setups including dynamic load controllers and communicating devices. The secondary data on manufacturing automation network properties are used to build the mathematical network model, which is simulated using mathematical simulation software MATLAB with its extension Simulink. Secondary data on control methods form the basis for the design of the controller to extend the simulation to simulate closed-loop network control with a single automation controller containing multiple talkers that address multiple listeners. Control engineering optimisation techniques according to Ziegler-Nichols or Chien-Hrones-Reswick (Normey-Rico & Camacho, 2007) are used to optimise the controller parameters. Building on these results, the model simulation is further extended to cover multiple automation controllers with multiple listeners network setups. Furthermore, the bridges, talkers, listeners, and the new control algorithm are implemented in software to be integrated into network simulation tools. The complete network simulation is

achieved by the application of the network simulation software ns-3 (ns-3, 2023; Wehrle et al., 2010), which provides additional primary data for data analysis.

3.2.1 Simulation of the Control Circuit

To determine the characteristics of automation communication networks as dynamic systems, it is necessary to first develop a model of a segment of a communication network.

This research project focuses on manufacturing automation communication networks based on layer 2 technology which is typical for field- and controller-level networks. It can be easily extended to layer 3 network technology. Such a layer 2 automation communication network consists of two types of network devices:

1. Communication end stations taking the talker role and/or the listener role.
2. A number of Layer 2 network bridges (also called network switches).

Figure 3.2 shows a general example of an arbitrary network segment.

Figure 3.2: An arbitrary network segment between two end stations

Any data path between two communicating end stations consists of one to *n* network bridges and their connecting links. The special case that two end stations are directly connected via only one link is not regarded, as it is out of question, due to missing interfering data traffic being inserted into the path via bridges along the path. The data path is identical to the plant of the control system. Its properties depend on the properties of the bridges and links.

A data transport link is assumed to be realised on wire-bound data transport, for example, standard Ethernet cables for 100 Mbit, 1 Gbit, or 10 Gbit bandwidth. One

Gbit dominates contemporary TSN MAN and is the focus of this thesis. Higher bandwidth applications, such as 40 Gbit or 100 Gbit, are not yet present in layer 2 automation network setups and are beyond the scope of this research project. The calculation of the different path delays for the different TSN MAN traffic shapers and schedulers leads to different dead time elements in the control plant and thus different control particularities.

For the simulation of the control loop, the MATLAB/Simulink tool, which is widely used in the academic and research domains of control engineering (Chaturvedi, 2017), is selected. The control plant and controller design are introduced into the tool using block diagrams. These can be selected from a variety of block element libraries, such as mathematical operations, logical operations, bit operations, different input signal forms, lookup tables, and output visualization tools, such as value displays or scopes, to name only some of them. Controller designs can be easily modified and optimised, and the results can be directly verified and documented using output visualization facilities. Thus, the control method can be optimised.

Step response diagrams provide quality statements regarding the settling time, overshot tolerances, and possible oscillations of the control circuits. Nyquist diagrams and calculations relating dead time and lag times provide insight into robustness and stability of the control circuits (Goodwin et al., 2001; Normey-Rico & Camacho, 2007).

3.2.2 Simulation of the Network

To gain knowledge on how a load distribution control algorithm will perform in a real communication network, the next step is to simulate the complete automation communication network.

The network simulation allows a detailed analysis of how the network control algorithm works under the variation of certain preconditions (Wehrle et al., 2010), such as:

- the amount of available network paths from talkers to listeners.

- the number of talkers and listeners from 1 to n, where n is a reasonable number of typical maximum number of network participants in a selected relevant use case.
- the number of bridges in the network.
- the type of bridges, that is, the technology of scheduling and queuing of the data traffic used inside the bridges.
- types of network data traffic such as data bursts, cyclic data, and synchronous or asynchronous data input.

There are a variety of network simulation tools available, for example:

1. the ns (network simulator) series with the latest version ns-3 (ns-3, 2023; Wehrle et al., 2010). This simulator is primarily intended for the research and education community, and is available under the GNU licensing model, that is, it is free to use, but contributions or changes to it must, in the same way, be made publicly available. It provides interfaces for statistical analysis tools and visualisation of simulation data.
2. The NetSim network simulation and emulation tool is a commercial tool that provides support particularly for wireless technology and layer 3 support.
3. The OMNeT++ simulation environment (Wehrle et al., 2010). OMNeT++ is also free to use in the research and education community but provides with OMNEST, also a commercial version.

For this research ns-3 is selected (ns-3, 2023) as network simulation tool for the following reasons:

1. It can simulate complete layer 2 networks and layer 2 network protocols (Henderson & Imputato, 2023).
2. Own protocol implementations or application implementations, such as load controllers, can be integrated as C++ source code.
3. It is widely used in the research community and a large number of protocol libraries and example codes which can be used to design tailored protocols and applications (Henderson & Imputato, 2023).

The ns-3 network simulator platform provides generic infrastructure for creating, running and evaluating network simulations for wired or wireless networks, sensor networks, automation networks, and many other applications. For this thesis it is used in a Linux environment.

The bridges and end stations can be instantiated from standard C++ libraries within the platform. These can be combined to form any topology. Special functionality for bridges and end stations can be introduced by modifying the present class source or creating modified classes. Additional source code for the controller and bridge functionality such as flow controller, distribution controller, rolling mean throughput measurement, throughput feedback processing, and load balancing application have been implemented and integrated into the simulation environment. The talkers can be operated with different traffic forms such as cyclic or burst traffic, and various cycle times. In addition, interfering traffic can be inserted into network path bridges along the network path. The resulting bandwidth consumption on the single bridges output ports can be recorded and analysed in terms of the performance of the control algorithm in “virtual real” networks.

Any network data at any point in time during the simulation time can be exported into data files, including a time stamp with a resolution of one nano second. Thus, there are no limitations in analysing network events such as the transmission or receptions of frames, dedicated runtime calculations, or even events between protocol stack layers. The data files are then processed to present the data via step response diagrams and calculation result diagrams using the Gnuplot plotting tool.

Thus, the network simulation platform offers a huge space of investigation possibilities for analysing the research problem and answering the research questions.

3.3 Methods for Data Analysis

Secondary data is used for the analysis and evaluation of current load control methods in ISP and campus networks. The research contributions to network load control are categorised by the applied types of control methods. The control application areas of various control methods are correlated to the network properties to which they have been applied to. Exploratory data analysis is applied by evaluating these control methods regarding their aptitude to achieve adequate load distribution control results in manufacturing automation networks. Therefore, characteristic manufacturing automation network properties are identified and correlated with the strengths and weaknesses of various control methods to extract and possibly improve the best suited control method or to alternatively propose a better-suited control method.

The primary quantitative data resulting from the simulation of the network model controlled by the selected suitable closed-loop controller design, is analysed with respect to the achievable results of quality criteria such as stability, resilience, and reaction speed. Confirmatory data analysis is applied to compare the simulations outcomes of the optimum control design, which was derived from the exploratory data analysis of the secondary data collection, with the ideal behaviour of the quality criteria.

3.4 Ethical Issues

This research will be conducted in accordance with the University of Gloucestershire Handbook of Principles and Procedures on Research Ethics (University of Gloucestershire, 2014). The principles of informed consent, anonymity, and confidentiality will be observed. In particular, references to examples of manufacturing automation use cases and solutions and to example network topologies, shall be of general types commonly known and applied by the industrial manufacturing community, avoiding examples of any proprietary vendor-specific solutions. Should it be necessary to involve any other organisations' confidential data, it shall only happen with the informed consent of

these organisations.

3.5 Chapter Summary

The research project is embedded in the positivism research paradigm as it produced quantitative data provided by modelling and simulation of dynamic load control solutions for TSN MAN.

Secondary data has been obtained from the literature review of load-balancing solutions in ISP networks, campus networks, mobile access networks, and cloud computing networks. The review of control theory and network theory literature, relevant IEEE, IEC and IETF standards, and relevant automation use cases has provided further secondary data.

Primary data has been obtained by building a mathematical network model for different TSN MAN traffic shapers and schedulers, which has been simulated using the mathematical simulation software MATLAB with its control engineering extension Simulink. Step response and Nyquist diagrams have presented the data and have served for the analysis.

A new dedicated control method, optimised for a TSN MAN has been designed and presented. Network simulations with ns-3 for a single AC have been used for confirming the improvements of load distribution convergence time.

The new dedicated control method has been extended for application in multiple AC TSN MAN under the influence of different TSN traffic shapers and schedulers. Two solutions have been proposed: one that is suitable for EST, CQF, and ATS without mutual controller dependency, and one for SPQ, which shows mutual controller dependency. Performance evaluations have been made comparing the solutions.

Chapter 4 The Influences of TSN MAN Properties on Load Distribution Control

4.1 Introduction

Within the area of campus networks, ISP networks, or mobile networks, load control concepts, such as Traffic Engineering (TE), Network Load Distribution or Load Balancing (NLB), have been applied for a few years, as shown in the literature review. These concepts are typically applied to networks based on OSI Layer 3, also known as routed networks. The advantages of NLB are well known, and researchers have conducted valuable work on this topic. Unlike in OSI Layer 3 networks, in OSI Layer 2 networks, and here especially in MAN and TSN MAN, NLB concepts have thus far not been investigated to that extent as seen in the OSI Layer 3 area. To design optimum control methods for the TSN MAN, among other requirements, it is also necessary to clarify the special properties of the Layer 2 TSN MAN compared to the solutions seen so far in Layer 3 general networks.

Because of their high reliability, both redundant end stations and redundant communication paths have also been gradually introduced into MAN solutions in recent years. New network standards as defined by the TSN project promote the use of multiple paths. Simultaneously, they defined various new functions to achieve highly efficient data transport. Redundant paths are currently used nearly exclusively for media redundancy. They serve either as standby paths or doubly transport data for seamless redundancy for failure protection between the data source and sink. However, they also enable the application of NLB concepts for automation networks to use these networks more efficiently.

In the routed ISP, campus, or mobile access networks, there are different load balancing concepts known, which can be categorized into three main concepts, namely “oblivious routing,” “predictive routing,” and “dynamic routing.” The latter is sometimes also named “adaptive routing.” Network load balancing concepts based on dynamic or adaptive routing use closed-loop control to control the bandwidth usage of network data paths. This is the controlled system output

required to achieve a homogenous load distribution among the available network paths from a data source to a data sink.

MAN differ in many ways from ISP networks or campus networks though, as they are mainly based on OSI Layer 2 rather than OSI Layer 3. In addition, the data traffic is of different type, in particular:

- data frames are typically smaller,
- the data transport intervals are much faster,
- the data traffic is typically generated in bursts instead of a homogenous data distribution over time,
- and it is, in certain borders, more predictable.

In addition, it is common that a higher number of automation controllers (AC) and end stations (ES), each of which can in turn host a multiplicity of talkers and listeners, share the same network. Beyond that, multiple ACs within the same ring pose a particular challenge in the pursuit of a distributed LDC concept. They can influence each other's load distribution calculation results via possible sections of common paths along their paths from the data source to the data sink.

A central question is whether load distribution in the OSI Layer 2 MAN is even possible and sensible. To answer this, MANs must be analysed regarding the following:

1. Should the control concept be based on central control or a number of distributed controllers?
2. Which network topologies are relevant?
3. Which bridging standard IEEE 802.1Q features are favourably used?
4. To what type of data traffic can it be applied?
5. How can the network use be controlled as a plant?
6. Which influence do the different TSN traffic shapers and schedulers have on plant properties?
7. Which influence have the automation applications properties?
8. How can LDC coexist with stream bandwidth reservation?
9. What influence do other TSN features have?

10. What influence do network errors have?

These questions are analysed and answered in the following sections.

4.2 Central or Distributed Control Concept

One of the first dedicated standards for TSN was IEEE 802.1Qcc (2018), which defined extensions for stream reservations. This was mainly aimed at audio/video bridging (AVB) applications, rather than industrial automation applications. However, two basic models for network configuration have already been defined. The central approach resides in a central network controller (CNC), and a distributed network configuration is located at the end stations. The central approach was not least the result of extensive research work in recent years in the area of SDN networks.

These two models are also important for MAN today. Here, it is necessary in the same way to decide whether the network load control should be positioned centrally or distributed. Figure 4.1 depicts these two basic concepts.

With Central Load Distribution Control (CLDC), a central control instance located on either a workstation or a single AC is responsible for a constantly optimised load distribution in the entire network domain. With Distributed Load Distribution Control (DLDC), the load control is located on several distributed ACs, each responsible for its own traffic distribution.

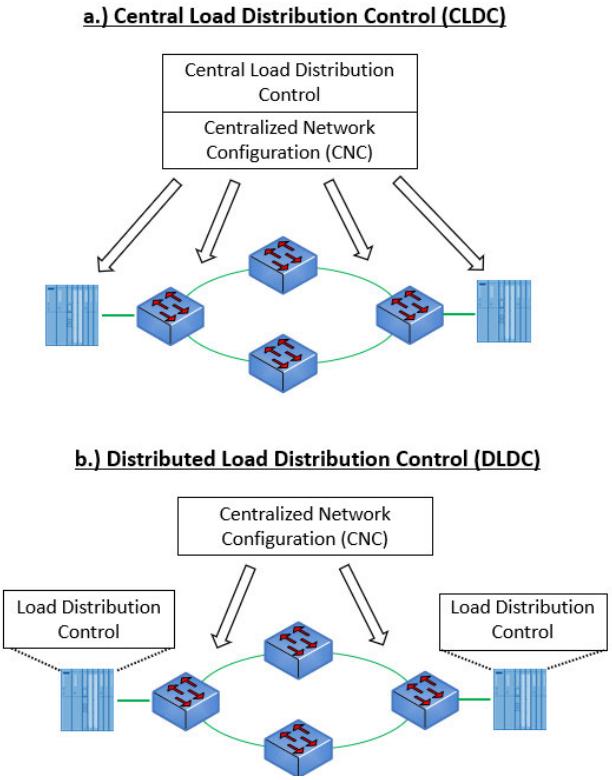


Figure 4.1: Models of a.) Central Load Distribution Control (CLDC) and b.) Distributed Load Distribution Control (DLDC).

There are several pros and cons that speak for one or the other concept. In practice, only for the network configuration and path control functionality, these two different approaches are already visible in the automation industry field with the establishment of the Avnu (2023) and LNI4.0 (2023) organisations, which both promote the application of TSN automation networks. Avnu favours central network traffic engineering and configuration, whereas the LNI favoured concept is distributed. However, both organisations currently have no activities on dynamic LDC. Network configuration is not the focus of this research, and it is assumed to be in a central instance in the form of a CNC, as shown in Figure 4.1.

Considering the advantages and disadvantages of a central and distributed LDC solution, the following considerations are important.

1. **Creation of additional traffic bottlenecks:** A central network controller would always need to be present on a dedicated powerful machine to calculate the best traffic distribution of all present traffic, as it needs to react quickly to possible traffic changes. This implies a constant polling of the measured load

values from each network node. This can be problematic, especially with larger networks causing additional traffic loads towards the CNC, being situated outside the ring connected via an uplink. The distributed solution with load controllers on various ACs better distributes the load measurement traffic. A recommended solution to ease the load caused by the exchange of load measurement and load scheduling values for both distributed and central control is to work with continuously circling summation frames. These would contain the values from and for several bridges and ACs within a single long frame.

2. **Susceptibility to errors:** With the distributed solution, the failure of one AC will not spoil the entire network load control concept. In contrast, this would be the case with central network controller loss.
3. **Network reconfigurations:** Typically, the goal of a central solution is to optimally configure the entire network. This includes the traffic distribution with a minimum delay for single frames to reach optimum results for applications. Depending on the objective for the quality of traffic distribution and the allowed delays, this could mean constant adjustment of gating windows, frame transmission slots, or even frame transmission points in time. The dynamic addition or removal of network participants can then easily spoil the previous optimisation result, requiring recalculation and reconfiguration of parts of the network or of the complete network. This can take a few hours for several hundreds of streams (Nayak, 2018). Furthermore, within the MAN, a particular feature is the hot plugging of the hardware. An example is the "Configuration in Run" (CiR) feature, as part of the PROFINET (IEC 61158-5-10, 2023) protocol. This allows the extension of PLCs or decentral peripheral I/O stations with additional I/O cards during runtime without stopping other applications running on that device. This is problematic for a centrally controlled load distribution, as it would mean disturbing the communication of other already running applications. The disturbances can be caused by the re-establishment of better paths and resource reservations, which are often accompanied by short communication interruptions or load distribution changes. In contrast, the distributed load distribution calculation and path and

reservation establishments for one AC typically do not affect the previous settings for other ACs control optimisation results, thereby avoiding these disturbances. This is especially true for the DLDC solutions provided in Section 7.4, where the ACs mutual influence is decoupled by time or by control sovereignty passing.

4. **Optimum network load distribution:** A central load distribution control solution has access to all network information and can use this to determine an optimal traffic distribution. The distributed solution usually cannot work with all network information and has no possibility to influence other ACs traffic distributions.
5. **Mutual controller influence:** With the distributed solution it will be the case that the load distribution changes caused by one controller have influences on other controller's calculation results. This is the case with common paths from different ACs to end stations. This is particularly true in ring topologies, which are the prevailing network topologies in MANs. The consequence can be load oscillations that are difficult to control. Therefore, the goal for a distributed solution must be a decoupling of the controllers or a solution that can cope with these mutual dependencies. Solutions to this problem are discussed and presented in Section 7.4.

The load distribution control method to be selected, the central or the distributed, must be decided by the preconditions of the applications performance requirements, the network properties, and possibly by already present solutions for network configuration, path control, and resource reservations. For the MANs on which this thesis is based, the distributed approach is selected for its better dynamic properties, which are the most important for the objective of this study.

4.3 Relevant Network Topologies

In factory automation applications, communication networks are typically based on OSI Layer 2 technology using switching. To date, these communication connections have been established redundantly primarily for fail-safety rather than for load-sharing purposes. Redundant connections require path-changing

redundancy protocols such as RSTP/MSTP or MRP for non-seamless traffic. An alternative is seamless redundancy protocols, such as PRP, HSR, or FRER, for seamless (doubly sent) traffic.

To achieve redundant connections with a minimum of wiring effort, the ring topology has become the prevalent topology in redundant industrial automation networks (IEC 62439-2, 2021; IEC 62439-3, 2021; IEC/IEEE 60802, 2018). Figure 1.1 shows a typical industrial automation network setup, where several field level rings are redundantly coupled to a controller level ring which again is redundantly coupled to a higher-level IT or OT network. Thus, the ring and redundantly coupled ring network topologies are relevant topologies in up-to-date automation networks as the focus of this thesis. Figure 4.2 shows their core structures.

Controller-level ring 1 usually contains a variety of higher-level automation controllers (AC1 to AC3), such as Programmable Logic Controllers (PLC) or Motion Controllers (MC). An AC can be attached singly via a separate link, such as AC1, or it can be integrated into the ring (AC2 to AC6) if it is a bridged end station, that is, it contains its own 3-port bridge. In this case, one port of the internal bridge is connected to the end station host, and the other two ports are the ring ports of the end stations. A field-level ring typically consists of only one AC that controls a variety of automation devices, such as drives, sensors, actors, or distributed peripherals, providing digital and analog inputs and outputs.

Interfering communication enters a ring, usually at the redundant coupling between rings (c1-c3). This can be data exchange with a local supervising controller, storage, cloud connection, edge application, edge management system for monitoring and diagnosis, device application updates, or device firmware updates. This inter-ring communication is often the reason for an additional asymmetric load in certain ring elements. For ring LDC, this traffic represents exogenous traffic, whereas the traffic caused by end stations directly connected to the ring is endogenous traffic from the ring’s point of view.

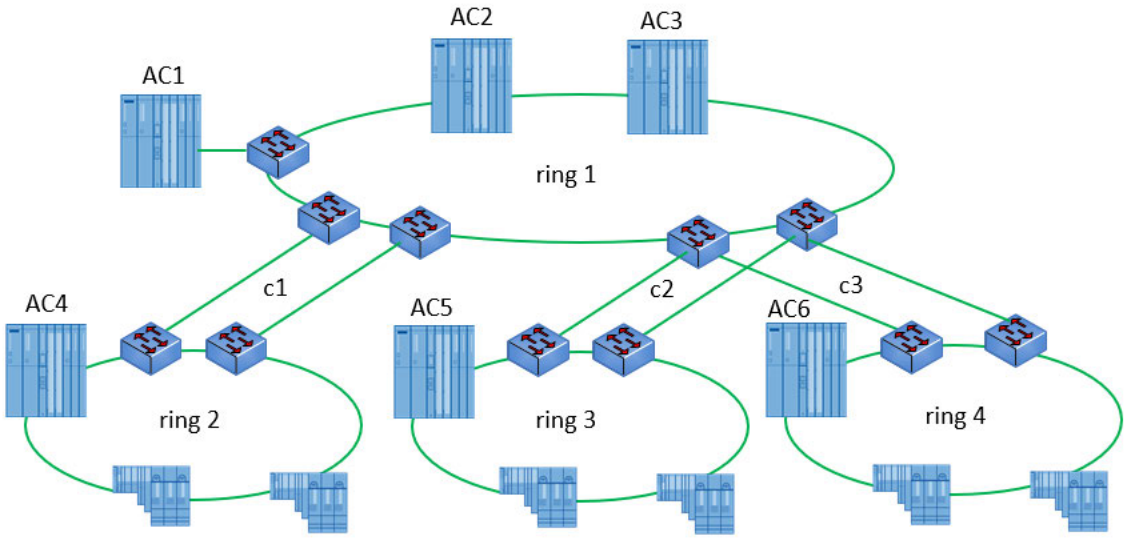


Figure 4.2: Rings and redundantly coupled rings

It is obvious to primarily control the load distribution of the endogenous traffic in the rings, as these transport a major part of communication. Thus, it is recommended, and the principle of this thesis, to handle each ring as a separate LDC domain.

Ring and redundantly interconnected rings are thus the relevant topologies when investigating load distribution in MAN. The primary goal is thereby the load distribution control within each ring rather than inter-ring load control, which could be relevant for a further step for a subsequent research task. Partly meshed or fully meshed networks, as they typically appear at the redundant access switches towards the higher-level network parts of the superordinated OT or IT plant or campus infrastructure, are not the focus here.

4.4 Path Control and Load Distribution Control Location

Layer 2 Ethernet networks, according to IEEE 802.1Q (2022) provide different mechanisms to use different data paths, which are an inevitable prerequisite for load distribution. The physical network is provided by bridges and their connecting links, that is, wired or wireless connections between bridges. Logical networks can be set up on top of the physical network by configuring VLANs. The goal of load

distribution is to provide at least two separate paths from each source bridge to all other bridges in the load distribution domain. The bridging standard (IEEE 802.1Q, 2022) basically provides two methods for achieving such logically separated trees within a physical network. These can be categorized as follows:

- administrator influenced path control approaches during network commissioning, and
- automated path control approaches during network startup.

Administrator influenced path control approaches:

1. With **MSTP**, each source edge bridge providing links to talkers can be configured as the root bridge of two MSTI in the MSTP region. The network configuration is responsible for configuring redundant paths rooted at the talker edge bridge. This can be achieved by assigning different path costs for single links, thus influencing the path setup, as the MSTP selects the path with the least cost. MSTP is usually combined with destination path selection via FDB learning. Therefore, the non-stream CD or BE would be adequate traffic to be load-controlled over these trees. The disadvantage of the MSTP is that the reconfiguration time after a network change is in the range of a few seconds, which is a relatively long time, especially for automation networks.
2. **SPB** protocols SPBV and SPBM are successor solutions for STP protocols. They are based on the link state protocol IS-IS, which provides multipath routing over multiple shortest paths. Unlike the MSTP, it uses the shortest path from a source bridge to a target bridge without having to follow laboriously a tree over a root bridge. The SPBV features different VLAN Identifiers (VID) assignments for the shortest paths. The SPBM uses an additional backbone MAC address for each edge bridge to identify the paths. From the viewpoint of media redundancy, SPB protocols have a much faster convergence time in the range of a few 100 ms. The precise values depend on the actual hardware and software design. In addition, the SPB can handle up to approximately 1000 bridges, whereas the MSTP is typically reduced to network diameters lower than seven, which results in a maximum number of bridges of less than 50. SPB

is mainly applied to data center solutions with closely meshed networks. However, it is rarely seen in automation networks at the controller level and much less so at the field level. As with MSTP controlled paths, non-stream CD or BE would be appropriate traffic to transport over SPB paths.

Comparing MSTP and SPB, it is logical to recommend SPB, because of its faster reconfiguration time, the shorter paths, and the higher possible number of nodes. This applies above all to arbitrarily meshed networks, where SPB then follows Shortest Path trees (SPT) which are marked by different VIDs. However, as the ring topology forms the basis for MAN, the possibility of using a shorter path with SPB is not given. This special case is also the reason why it has no effect on the LDC whether the paths are provided by MSTP or SPB, they will be the same. Furthermore, the automation nodes within MANs typically do not provide SPB functionality. These facts reduce the choice for non-stream CD and BE paths in current MAN designs to the MSTP, in which two redundant MSTIs are to be set up in the two ring directions from each edge bridge, in parallel to stream VLANs. An LDC entity can be located in either an end station connected to the ring bridge or in the bridge itself. Both solutions are possible when using MSTP for path control.

Automated path control approach:

As an alternative to a manually influenced configuration, different redundant paths for LDC can also be found using different ISIS-PCR algorithms, such as an Explicit Equal Cost Tree (ECT) algorithm or a Maximally Redundant Tree (MRT) algorithm (IEEE 802.1Q, 2022). The actual path-finding algorithms are calculated in a Path Computation Element (PCE), which is located either in an end station or in a bridge, to achieve the necessary path configurations in all bridges of the SPT domain. If the PCE is in the end station, it corresponds to a Path Computation Agent (PCA) in the connected edge bridge. Each SPT bridge provides a Bridge Local Computation Engine (BLCE) to cooperate with the PCE or PCE/PCA. Thus, an important question in connection with the design of a distributed load distribution mechanism within an OSI Layer 2 network is, where the PCE shall be located. In an end station or in a bridge? In both cases, an integrated PCE/LDC solution, would be advantageous.

Figure 4.3 depicts the possibilities for the location of PCEs and LDC entities for singly attached end stations. These can then be present either in only one representation as a CLDC, or multiply instantiated in the DLDC approach.

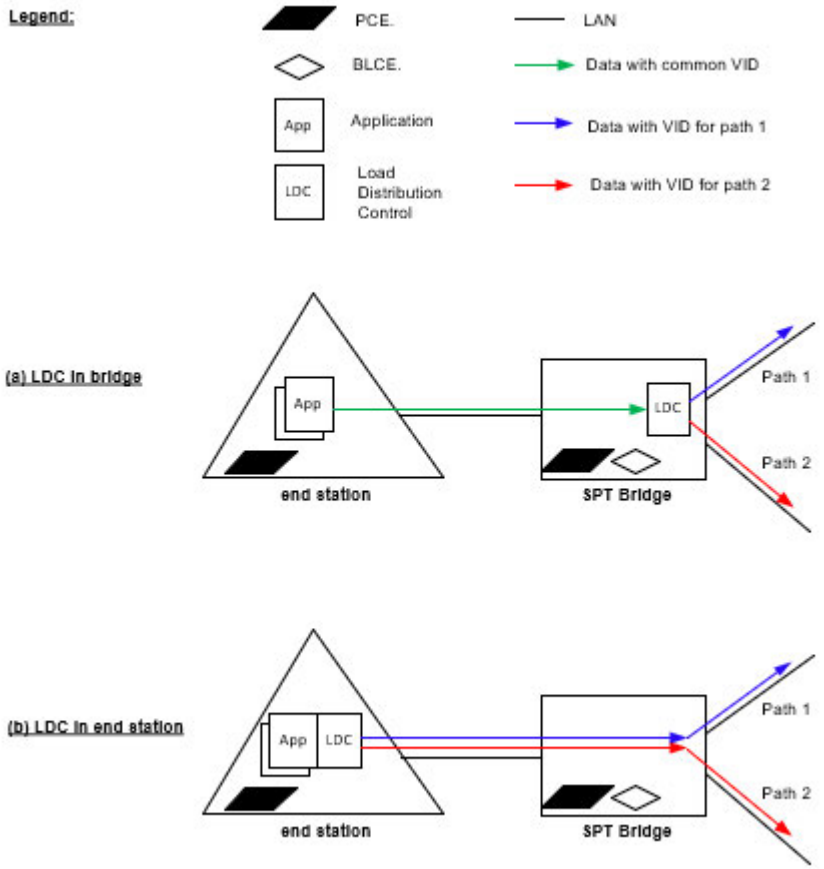


Figure 4.3: Location possibilities of PCE and LDC entities

There are two possibilities for the LDC to be located:

1. As shown in Figure 4.3 (a), the LDC can be located in the bridge and assigns certain arriving data traffic with a certain ingress VID onto either path 1 or path 2, depending on the current load distribution control result. As an alternative, it could implement an oblivious or round-robin routing method by forwarding frames to both paths in alternation, which in sum, under the participation of all bridges, achieves a better load distribution.
2. Alternatively, as shown in Figure 4.3 (b), the LDC can be located in an end station where it directly calculates the load distribution of the data generated

by different applications.

In both cases, the traffic distribution is achieved by assigning two VID for the two possible paths. The data are then forwarded accordingly in the edge bridge because of the Forwarding Database Identifier (FID) configuration achieved by the PCE or PCE/PCA combination in cooperation with the BLCEs.

For the first possibility, the LDC location in the bridge, the appearance of a load distribution mechanism is expected to be rather of the nature of a decision algorithm whether to forward onto one of either paths or on both paths in alternation, according to simple and low-calculation-effort mechanisms. The reason is that a bridge, whose original task is to filter and forward incoming data to other ports, is not the right system for the calculation of rather complex load control algorithms.

The second possibility, the LDC end station location, is typically the case for an AC connected to a ring bridge. This is the right selection to calculate the control algorithm if it is an influential AC that provides both a reasonable amount of data to be subjected to load control and sufficient hardware resources to calculate the control algorithms. As described in the previous subsection, the AC can also be a doubly attached end station located directly in the ring and has an integrated bridge function.

4.5 Eligible Traffic Classes

Further protocol and hardware design decisions must be made for different data traffic types in a MAN.

The IEC/IEEE industrial automation TSN profile (IEC/IEEE 60802, 2018) further classifies the automation data traffic as listed and extended in Table 4.1.

Table 4.1: Industrial automation traffic types

Traffic type name	Periodic (cyclic)/ sporadic	Examples
isochronous cyclic real-time	periodic	Isochronous control data (I-CD)
cyclic real-time	periodic	Non-Isochronous control data (NI-CD)
network control	sporadic	Network administration
audio/video	periodic	Visual monitoring traffic
brownfield	periodic	Non-Isochronous non-TSN cyclic control data of a neighbor machine or network or devices.
alarms/events	sporadic	Device or network alarms
internal/pass-through	sporadic	Clock synchronization, media redundancy etc.
best effort	sporadic	Firmware/application updates
best effort	periodic	Continuous cloud or edge connection data

NI-CD and I-CD are expected to contribute the most to the traffic load because of their cyclic occurrence. They therefore deserve special attention for the design of an LDC for TSN MAN.

Figure 4.4 shows the classification decision stages. The different traffic types according to Table 4.1 must be classified according to these. The design properties shown are not a complete list of all possibilities but consist only of those that are relevant for multiple path networks, which is a compulsory precondition for load distribution.

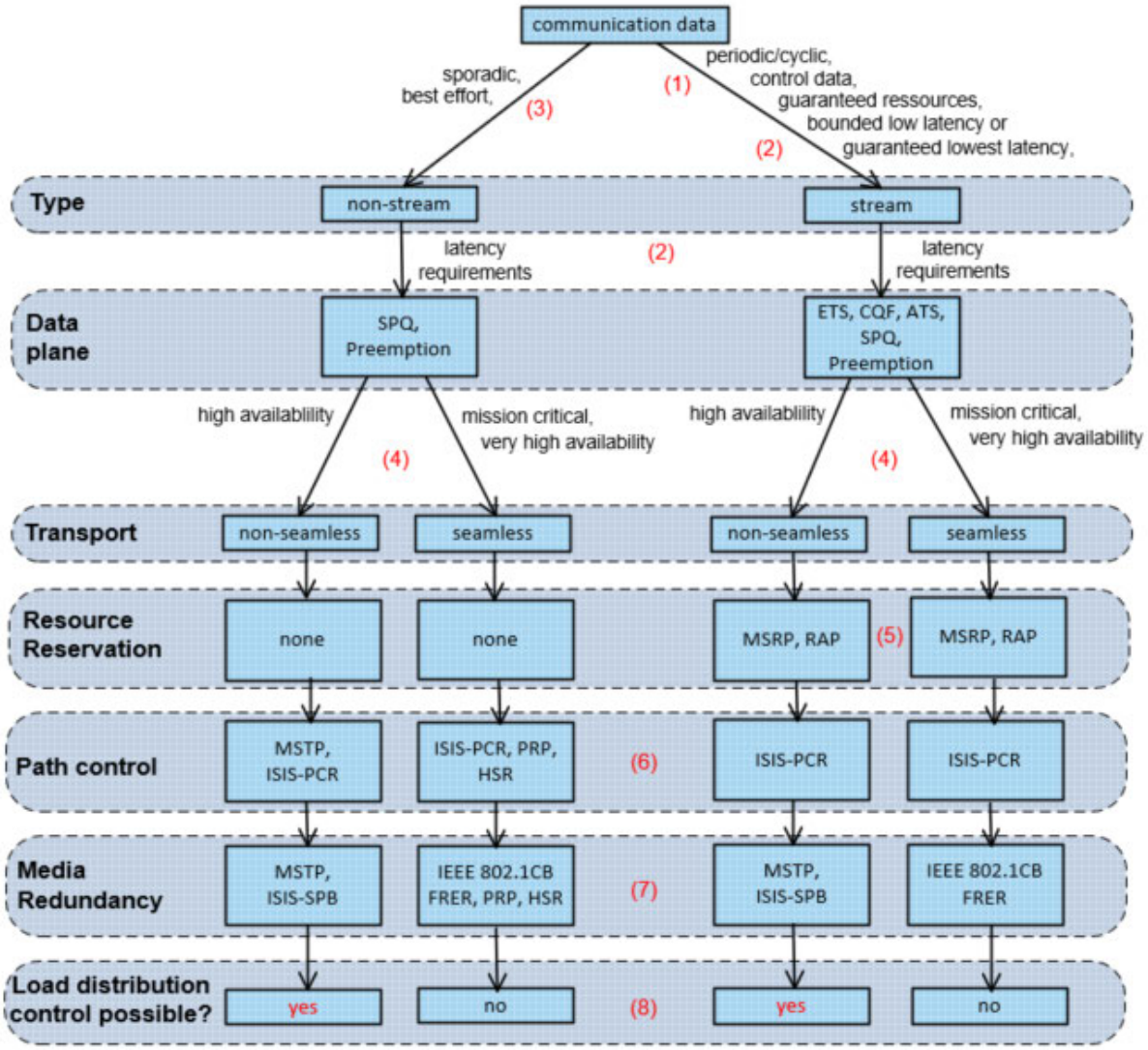


Figure 4.4: Decision criteria for data traffic classification

Beginning from the top of the classification process, the communication data must be classified as follows (the bullet numbers refer to step numbers (x) in Figure 4.4):

(1): Periodically or cyclically flowing data, such as isochronous CD (I-CD) or non-isochronous (NI-CD), are best designed as streams that can be subject to resource (bandwidth) reservation.

(2): Streams are also the recommended traffic type if the data are transported by the lowest latency mechanisms such as EST or high priority SPQ with pre-emption, or if lower bounded latency shall be achieved with unsynchronised data using the CQF or ATS traffic shapers.

(3): Traffic of sporadic, that is, non-cyclical, character, or BE traffic with relaxed latency requirements are best packed into the group of non-streamed data with no reserved bandwidth or transmission time gates.

(4): The next crucial requirement is the so-called grace time of data flow interruption, which decides whether data is to be transported seamlessly or non-seamlessly. Mission critical data with very high availability requirements of only a few milliseconds of grace time need to be transported seamlessly. Traffic with lower availability requirements spanning from a few 10^{ths} of milliseconds to a few seconds can be transported singly or non-seamlessly.

(5): Resource reservation, that is, bandwidth reservation, is only provided for streams and can use the reservation protocols MSRP or RAP.

(6): To provide at least two logically separated network paths for load distribution or seamless data transport, ISIS-PCR is the path control protocol defined by IEEE 802.1Q (2022). For non-stream seamless data, paths can also be provided through the redundancy protocol HSR, which is defined for rings, or by PRP which bases on two physically completely separated networks. For non-stream, non-seamless data the MSTP or MRP would be an alternative.

(7): Media redundancy for seamless streams can be provided by FRER (IEEE 802.1CB, 2017). For non-seamless streams, switch-over redundancy in a multipath environment can be achieved using MSTP or ISIS-SPB. The same counts for non-seamless, non-stream traffic. Seamless non-stream media redundancy can also be achieved via PRP or HSR.

(8): Load distribution control is always possible for non-seamlessly transported data. For streams it can be combined with all available traffic shapers and with ISIS-PCR for path control and ISIS-SPB for media redundancy. For non-streams it can be combined with all SPQ priorities and preemption. For general meshed networks, IS-IS technology with ISIS-PCR for path control and ISIS-SPBV for media redundancy is recommended rather than MSTP because of ISIS-SPBV outstanding path selection effectivity and flexibility (Huawei, 2010). However, for MAN ring topologies MSTP is the better selection as outlined in Section 4.4.

The described classification procedure is, of course, only a best-practice procedure and there might be important reasons to deviate in one or the other cases from this. For example, sporadic traffic could be so important that it would be worthwhile to spend guaranteed bandwidth resources and guaranteed lowest latency, even though the reserved bandwidth is only used to a low extent owing to the sporadic character of the data.

Summarising this classification for CD, non-seamlessly transferred data are a possible candidate for load distribution. This is because the doubly transferred CD cannot be subject to load distribution because ingress limiting is not applicable for CD, only the redirection of traffic. In certain cases, Non-CD can also be transported seamlessly and can typically cope with further delays caused by load-controlled throughput reduction. Seamless and non-seamless I-CD and NI-CD in TSN networks are typically separated by VLANs. Seamless CD contributes to the basic load of non-load-controllable data. Non-seamless CD with higher bandwidth consumption are available for load control. Non-seamless CD with low bandwidth consumption, such as sensor data, are often not worthy of load control and contribute to the basic load of non-load-controllable data. On the other hand, it is just this data of lower cost sensors, which will be non-seamless, as otherwise a rather expensive redundancy box for seamless traffic integration must be donated for each sensor. If sensor data have a higher bandwidth consumption, an AC can include this traffic in the LDC by managing the send port at the remote sensor. See Subsection 7.4.3 for more details on this.

Streams are always transported using a Layer 2 group MAC destination address. Thus, they are multicast frames of type “1-to-*n.*” It is at reservation time and transmission time not certain to what extent one or the other path will be used, that is, at what distance from the source or talker edge bridge the last listener is or will finally be located. The original idea of multicast stream establishment stems from the AVB application (IEEE 802.1Qcc, 2018), in which any number of listeners of an audio or video stream published by one talker could consume the stream. However, within automation applications, multiple listeners are the exception. An AC addresses single devices with dedicated streams containing process data only

for this device, although it uses a stream for this. The process data streams in the opposite direction from the device to the PLC are typically of type 1:1. Therefore, the probability that a further listener joins the stream at a further end of the path at a later stage after the initial stream setup is very low within automation applications and results in fixed path lengths for the streams. In addition, to make use of the LDC for streams, the stream reservation strategy must follow certain rules. Refer to Section 4.8 for details on the influence of the reservation process. If these rules are respected, the non-seamless NI-CD and I-CD streams are suitable candidates for load distribution.

Non-CD traffic is mostly transported via layer 2 unicast MAC addresses, that is, there is only one target for the data in the network. This is then of the type "1-to-1." If this type of data is to be distribution load controlled, the load conditions of the fixed paths from the source to the target are relevant and must be considered. For the Non-CD multicast type, the insecurity factor is that it is not known where the last target on each path is located. Therefore, the actual length of the paths is typically unknown, and the maximum possible length of the paths must be considered to obtain the maximum load on each. Under these circumstances, non-seamless Non-CD are potential candidates, both as unicast and multicast frames, for load distribution control. The option of throughput reduction of seamlessly transported Non-CDs is not investigated in this thesis, as this concentrates on load distribution, not reduction.

Summing up on these evaluations, the suitability of the possible MAN traffic types from Table 4.1 are assessed and comprised in Table 4.2.

Table 4.2: Traffic types for load distribution

Traffic type name	Periodic (cyclic) or sporadic	classification	Recommended transport	Potential candidate for Load Distribution?	Comments
isochronous cyclic real-time	periodic	I-CD	stream	yes	If transported non-seamlessly.
cyclic real-time	periodic	NI-CD	stream	yes	If transported non-seamlessly.
network control	sporadic	Non-CD	non-stream	no	Too sporadic. Too little data.
audio/video	periodic	Non-CD	stream	yes	Usually, high bandwidth consumption also in the field level. Potential candidate.
brownfield	periodic	NI-CD	non-stream	yes	If transported non-seamlessly. Path decision in bridge.
alarms/events	sporadic	Non-CD	non-stream	no	Too sporadic. Too little data.
internal/pass-through	sporadic	Non-CD	non-stream	no	Too little data.
best effort	sporadic	Non-CD	non-stream	yes	If enough data to be worthwhile.
best effort	periodic	Non-CD	non-stream	yes	If enough data to be worthwhile.

This summary shows that both CD and Non-CD in both transport types of streams and non-streams are potential candidates for the load distribution. However, there may be reasons to exclude data from distribution control. The amount of data of a talker can be too small to be a worthwhile application for load control. Another reason can be that the device hosting the talker can be of too low calculation performance to operate a load control, or the devices firmware may not be adaptable to host a load controller. This data is henceforth referred to as granular data. A further reason can be that data shall use a fixed shortest path to achieve a minimum latency. Figure 4.5 comprises these coexistence relations.

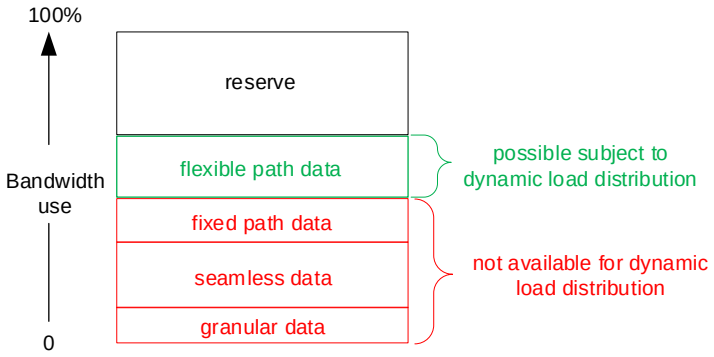


Figure 4.5: The coexistence of load-controllable data and non-load-controllable data on a network path

Therefore, the data that are not to be transported seamlessly, and are not bound to a fixed path, and are not granular data, can be load-controlled data.

Depending on the traffic types present in the MAN, different strategies can be used to achieve the load distribution. This will always be a compromise between the effort for the LDC and the quality of the distribution homogeneity. In the planning and network configuration phase, the initial distribution is optimised via traffic engineering as described in the previous sections. As a reaction to unplanned dynamic traffic imbalances, dynamic control can react with only a CD load shift if it provides sufficient distributable bandwidth. A further alternative could be to react only with CD with a certain minimum contribution. Or it could be necessary to make also Non-CD or BE part of the LDC. All combinations are possible.

4.6 Control Aspects

Communication networks for automation plants are typically planned to be within well-defined limits. Furthermore, they experience rather limited changes during runtime in terms of both the number of network participants and the participant's amount of data transfer. Therefore, it must be assumed that deviating control methods will be applied in automation network load control compared to those used in networks containing more uncertainty. Such networks would be Layer 3 ISP networks, campus networks, or mobile network access networks. Typically, these must cope with large amounts of un-plannable communication traffic.

Control methods for these include linear control, stochastic network load control (Neely et al., 2008), or control using fuzzy control or neural networks (Bolla et al., 1998; Kaszkurewicz, 2010; Wang & Hung, 2012). A comparison of their application fields is presented in Section 6.3.

The extent and layout of the automation network and its communication partners and their communication load requirements are largely determined at the time of network design and deployment. This suggests using the following load distribution design principles in MAN:

1. To apply traffic engineering in the network design and setup phase, that is, to identify the different traffic types and their transport requirements as specified in Section 4.5, with the goal of achieving homogenous network use already at network startup.
2. To apply dynamic or adaptive traffic load control at runtime for certain traffic to react to unpredictable network traffic changes.

For the second step covering dynamic traffic load control, the applied network control concept can either be based on a central approach, where all network configuration intelligence is located within a central network controller (CNC), or a distributed approach. This has been discussed in Section 4.2, where the decision has been in favour of the distributed concept. Its advantage is, that the network adapts more easily to new network users, that is, new end stations and their communication demands, while the network bridges provide configuration intelligence by providing path control and resource reservation facilities.

The early Zaki et al. (1996) concept of four steps for the purpose of distributed systems dynamic load balancing, as mentioned in Subsection 2.6.2, can also be applied in the transferred sense on network load distribution and constitutes the following steps.

1. Monitor path loads at each bridge port in the ring.
2. Calculate the maximum per path.
3. Calculate new favourable distributions.

4. Shift load from one path to the other via flow control.

The monitoring of the path loads or throughputs and the calculation of their maximum along a certain path is obviously best performed at the distribution controller site, which requires these values. However, the calculation of the sliding mean value over a configurable integration time is best performed in the bridges for each port. This relieves the network from extensive single-load value update data traffic from bridges at each measurement cycle.

The classification of LDC into flow control and distribution control, as introduced in Subsection 2.6.4, is also sensible for load control in automation networks. The data flow control is subsequential to the distribution control, as it controls the demand for an increase or decrease on a path, which is calculated by the distribution control. This is elaborated further in Chapter 5 . Fairness control is of secondary importance. The reason is that the proportion of timely rather uncritical data flows of Non-CD, whose throughputs could be evenly reduced, such as TCP/IP flows, is typically low in MAN. Instead, automation networks need to part the time-critical streams and non-streamed CD onto different paths without being allowed to reduce the overall throughput of the data. Therefore, congestion control, where ingress data are either dropped or the sender is informed to reduce the throughput, is not an option for CD automation data traffic.

To understand the process of data communication in automation networks from the viewpoint of load control, it is necessary to analyse automation communication more formally.

Figure 4.6 shows a section of an abstracted fully meshed automation network which is represented by the graph ***G** = (V, E)* with a set of vertices ***V**(G)* and a set of edges ***E**(G)*. Set ***V**(G)* represents the nodes v_i of the graph, representing either pure network switches or automation devices with integrated switches. The set ***E**(G)* with the edges e_{ij} represents the links between node v_i and node v_j. The number of nodes of the graph determines its order *n*. The number of edges connected to a node determines its degree *deg(v)*.

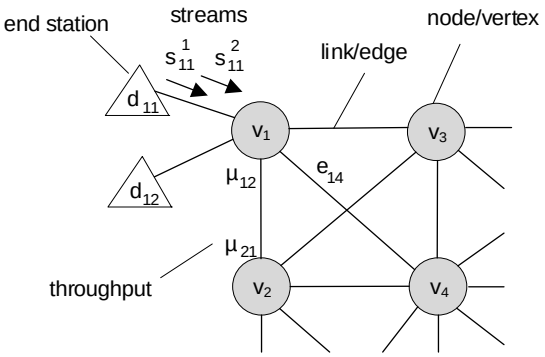


Figure 4.6: Abstracted TSN automation network

Let $\boldsymbol{D_i} = \{d_{i1}, \dots, d_{ij}\}$ be a set of devices (end stations) connected to node $v_i \in \boldsymbol{V} = \{v_1, \dots, v_n\}$, that is, a bridge. Let furthermore be $\boldsymbol{Ta_{ij}} = \{ta_{ij}^1, \dots, ta_{ij}^k\}$ a set of talkers within d_{ij} and let $\boldsymbol{Li_{ij}} = \{li_{ij}^1, \dots, li_{ij}^p\}$ be a set of listeners within d_{ij}. $\boldsymbol{Ta_{ij}}$ create a set of streams $\boldsymbol{S_{ij}} = \{s_{ij}^1, \dots, s_{ij}^q\}$ being sent to v_i. Per definition and in accordance with the definitions in IEEE 802.1Q (2022) one talker ta_{ij}^k issues only one stream s_{ij}^q that can be consumed by a set of listeners. Although the control data can be transported as streams or non-streams, for a more concise description, they are simply referred to as streams. The paths that the streams can take from a talker ta_{ij}^k to one or more listeners li_{ij}^p, located somewhere in the network, are derived from automation applications running in device d_{ij}. The sum of the directed streams on link e_{ij} create throughput μ_{ij} at the output port of node v_i. Each link e_{ij} provides two scalars of throughputs μ_{ij} and μ_{ji} which represent the current output data rates at node v_i in the direction of v_j and vice versa respectively. If no neighbor node exists for a certain port, no stream and throughput exist on this port either. Thus, the edges describing the throughputs are directed edges. The individual throughputs μ of the network can be formed as an instance $\boldsymbol{M}$ of a distance matrix of graph G:

$$\boldsymbol{M} = \begin{bmatrix} \mu_{11} & \cdots & \mu_{1n} \\ \vdots & \ddots & \vdots \\ \mu_{n1} & \cdots & \mu_{nn} \end{bmatrix} \tag{4-1}$$

where n is the order of the graph, which represents the number of nodes, that is, bridges and bridged end stations, within the automation network domain. If only a certain traffic class is in focus of the control, the preceding considerations can

also be made related to streams of a certain traffic class on link e_{ij} instead of all streams.

Automation applications with redundant networks are nearly without exception set up in ring topology as illustrated in Figure 4.7.

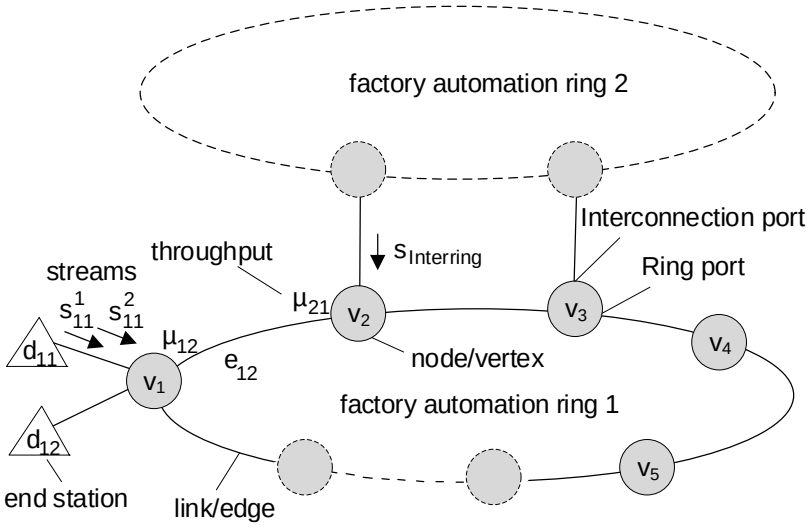


Figure 4.7: Automation ring graph

The throughput distance matrix ***M*** for a ring topology reduces to a doubly diagonal filled matrix, provided that the nodes of the ring are numbered clockwise or counterclockwise in succession. For example, ***M*** for a ring of five nodes results in:

$$\boldsymbol{M}_{ring} = \begin{bmatrix} 0 & \mu_{12} & 0 & 0 & 0 \\ \mu_{21} & 0 & \mu_{23} & 0 & 0 \\ 0 & \mu_{32} & 0 & \mu_{34} & 0 \\ 0 & 0 & \mu_{43} & 0 & \mu_{45} \\ 0 & 0 & 0 & \mu_{54} & 0 \end{bmatrix} \tag{4-2}$$

The ring nodes v_i provide the throughputs on their ring ports as feedback for flow control within the ring. Owing to various applications with talkers ta_{ij}^k connected to the ring nodes v_i and possible inter-ring communication $s_{interlink}$, the individual link throughputs along a path from a controller talker to listeners can be different. The inter-ring communication $s_{interlink}$ is from the viewpoint of the ring load distribution control an exogenous traffic. The streams of the end stations within the ring are endogenous traffic. For the purpose of this thesis, stream s needs to be assigned a further property which is its application cycle time. This requirement will be outlined in more detail in Section 4.7 and in the chapters to

follow. Therefore, the notation for a stream must be extended from s_{ij}^{q} to $s_{ij}^{q\alpha}$, with α ∈ ℕ, representing the application cycle time. This is typically coded as a 2^n ms value within MANs. Table 4.3 provides an overview of the applied notations.

Table 4.3: Notations

Symbol	Meaning
α	Application cycle class of a stream or throughput
$\boldsymbol{A}$	A set of application cycle classes used in the network domain
$\boldsymbol{AC}$	The set of all automation controllers in the network domain
γ	Index for the automation controllers (ACγ) in the network domain
cw	Clockwise direction
ccw	Counterclockwise direction
$\boldsymbol{D_i}$	A set of devices (end stations) being connected to a node v_i
d_{ij}	Device *j* connected to node v_i, part of $\boldsymbol{D_i}$
$\boldsymbol{E}$	A set of edges (links)
e_{ij}	Edge (link) from node *i* to node *j*
$\boldsymbol{V}$	A set of nodes
v_i	Node number *i*
$\boldsymbol{Ta}$	The set of all talkers in the network
$\boldsymbol{Ta_{ij}}$	A set of talkers within device *j* connected to node *i*
ta_{ij}^{k}	Talker k within device *j* connected to node *i*
$\boldsymbol{S_{ij}}$	A set of streams originating from device *j* connected to node *i*
s_{kl}^{q}	Stream *q* originating from device *l* connected to node *k*
$s_{kl}^{q\alpha}$	Stream *q* with application cycle α originating from device *l* connected to node *k.*
μ_{ij}	Throughput at node *i* in the direction of node *j*

The distribution control task to use one of either paths of the ring, results from the optimisation task to minimise the maximum throughput on the single links on the available paths:

$$\min \max_{i,j \in \boldsymbol{V}} \mu_{ij} \;\; ; \text{ Subject to: } \forall\, e \in \boldsymbol{E}(G) \tag{4-3}$$

This optimisation strategy to reduce a local maximum is named "**maximum-reduction**" method within this thesis.

Given that this research focuses on CD stream load distribution control, the throughput to be measured at any port thus comprises

$$\mu_{ij,CD} = \sum_{q \in \boldsymbol{Ta}} \sum_{\alpha \in \boldsymbol{A}} \mu_{ij,CD}\left(s^{q\alpha}\right) \; ; \; \boldsymbol{A}, \boldsymbol{Ta} \subseteq \mathbb{N}, \tag{4-4}$$

which represents the sum of all throughput contributions of all talkers CD streams from all possible end stations throughout the network, over all application cycles, within a certain CD traffic class, and at one output port. The pseudo code for finding and comparison of the throughput maxima of each of both ring directions output ports is outlined in Table 4.4. Its detailed structure is provided in Appendix 2.

A more ambitious goal would be the load distribution of a possibly optimum distribution or at least a distribution within a certain deviation tolerance. The latter strategy, which aims for a non-optimal but improved result within a certain deviation tolerance, is to be preferred. Finding a final optimum can be a very costly task both with regard to the calculation effort and calculation time. In most cases it is expected that an optimum distribution, which is in fact an equal load on all paths and links in all directions, will not be found. This is because randomly distributed talkers and listeners often communicate over hundreds and thousands of communication relations with each other, resulting from the various automation tasks in the network. An algorithm could try endlessly to find a more favourable load distribution if not stopped by another exit criterion. This optimsation task can thus be NP-hard. Therefore, the use of heuristics to improve load distributions is necessary.

Table 4.4: Pseudo code of algorithm for path throughput load maximum determination and comparison per node and path direction.

Algorithm: CollApp::Compare ()

This algorithm of the collection application in an AC finds and compares the throughput maxima of each of both ring directions output ports. It shall be calculated cyclically before each call of the distribution controller method in an AC.

```
Create three-dimensional array m_thp_array [] [] [] for storing load measurements of each
node, direction, and application cycle once at instantiation of this method;
Create "sum" and "max" variables;
sum = 0;
max = 0;
Receive each node's mean throughput load feedback-frames via interrupt in a receive
method in the background;
Store mean throughput loads from feedback-frames in m_thp_array per node, direction,
and application cycle;
Sum the individual application cycles throughput per node and direction to build the overall
bandwidth consumption (throughput) for CD:
For node i <= maximum number of nodes
{
  For direction j <= 2
    {
    For application cycle α <= maximum number of application cycles
      {
      sum = sum + m_thp_array [α] [j] [i] ;
      }
    Store sum over all application cycles at index SUMAPPIND:
    m_thp_array [SUMAPPSIND] [j] [i] = sum;
    }
}
sum = 0;
Find maximum per application cycles and over all application cycles in each direction and
store maxima of each direction in array:
For application cycle α <= SUMAPPSIND
{
  For direction j <= 2
    {
    For node i <= maximum number of nodes
      {
        If (m_thp_array [α] [j] [i] > max)
        {
         max = m_thp_array [α] [j] [i];
        }
      }
      Store max at index number of nodes NNODES:
      m_thp_array [α] [j] [NNODES]= max;
      max = 0;
    }
  Build the difference of the two path directions clockwise (cw) and counterclockwise (ccw)
  maxima and store the half of it in the array for the distribution controller which then tries
  to minimise the difference by load shifts.
  If ((m_thp_array [α] [CW] [NNODES] != 0) && (m_thp_array [α] [CCW] [NNODES] != 0))
  {
  m_thp_array [α] [CW] [NNODES + 1] = (m_thp_array [α] [CW] [NNODES] –
  m_thp_array  [α] [CCW] [NNODES])/2;
  }
}
```

The optimum-distribution goal can be defined as the least square optimisation goal between the actual load distribution and ideal load distribution. The best approximation of this ideal distribution represents the best possible network utilisation. An obvious way to achieve this is to optimise as follows:

$$\min \sum_{\substack{i,j=1 \\ i,j \in \boldsymbol{V}}}^{n} (\mu_{ij} - \mu_M)^2 \quad ; \text{Subject to: } \forall\, e \in \boldsymbol{E}(G) \tag{4-5}$$

where μ_{ij} is the link load in both directions of the ring according to Matrix $\boldsymbol{M}_{ring}$, (4-2), n is the number of nodes, and μ_M is the average load over both ring direction paths. The latter is defined as

$$\mu_M = \frac{\sum_{i,j=1}^{n} \mu_{ij}}{2n} \; ; \; n \in \mathbb{N} \, . \tag{4-6}$$

This optimisation strategy to find a possibly near-optimum distribution shall be referred to as "**optimum-distribution**" method within this thesis.

The optimal distribution method depends heavily on the application of a CLDC. However, since preference is given to DLDC in this study, the maximum reduction method is also the optimisation strategy chosen here.

The detailed control model for subsequent data flow control is analysed and applied to various TSN traffic shapers and schedulers in Chapter 5 . A load distribution control method optimised for MAN is proposed in Chapter 6 and extended to multiple ACs in Chapter 7 .

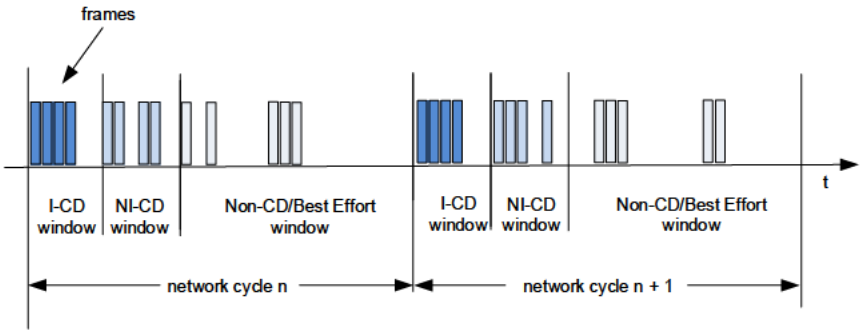
4.7 The Influence of the Automation Applications

Different automation applications usually demand a variety of application communication cycles with end stations. This results from the fact that control loops and other automation tasks are processed cyclically, each with their individual application cycle time. The application cycles are determined based on the individual automation application requirements. These are running on one or several ACs in the network, each having its own minimum communication cycle time with peripheral devices or other controllers. Communication typically occurs

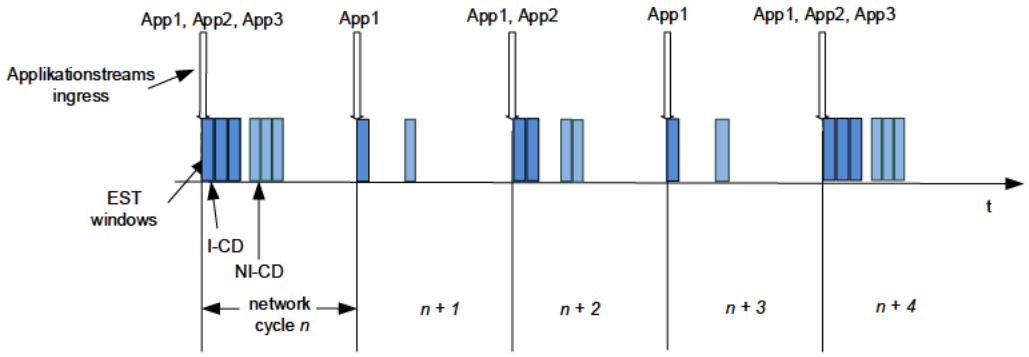
once at the start of an application cycle. For instance, a slow temperature controller might exchange the setpoint and actual value with an analog I/O card only every 500 ms, whereas a fast speed controller for motion control might need to exchange the setpoint and actual value in cycles of a few 10^{ths} of µs with a drive. Data exchange between the application on the AC and connected devices can be unidirectional or bidirectional, depending on the application. Data are typically exchanged once during the application cycle in each direction. Examples of unidirectional data exchange include the provision of reference values to an actuator or the provision of actual values from a sensor. Bidirectional data can be a closed-loop control that exchanges setpoints or reference values in one direction and actual values in the other direction once in the application cycle.

Besides the application cycle, nearly all types of TSN-based network types, except for SPQ, provide a network cycle time. This uses the timing information of the bridges to synchronise the data transport throughout the TSN domain. Thus, in particular, with EST, a minimum latency is achieved. The network cycle time is determined by the shortest application cycle in the network domain and may not be longer than this. Figure 4.8 shows how application cycles and network cycles are correlated within an automation network based on the EST traffic scheduler.

Figure 4.8 (a) shows a snapshot of two network cycles with EST windows for I-CD, NI-CD, and Non-CD or BE traffic classes. The I-CD data transport is fed in synchronised to the network cycle and its EST window start. It is transported immediately at the start of the window without any delay. NI-CD and Non-CD queues are emptied at the window start, and there might be additional unsynchronised data during the window duration time.



(a) Succession of network cycles with I-CD, NI-CD and Non-CD traffic in EST windows



(b) Succession of frame ingress from applications with different application cycles

Figure 4.8: Network cycle and application cycles

Figure 4.8 (b) shows a succession of five network cycles and three example applications App1, App2 and App3 feeding in I-CD streams and NI-CD streams of approximately the same size. App1 sends its data s_{ij}^{q1} with application cycles identical to the network cycle, $\alpha = T_{App\,1} = T_{Netw}$, App2 sends s_{ij}^{q2} with $\alpha = T_{App\,2} = 2T_{Netw}$, and App3 sends s_{ij}^{q3} with $\alpha = T_{App\,3} = 3T_{Netw}$. Thereby a network load pattern is generated which is repeated at a frequency of $1/T_{App\,3}$. Thus, the traffic pattern at a certain link is characterised by the slowest application sending data over this link. If there is no full control over each application data traffic path in the TSN MAN domain, it must be assumed that this pattern is repeated at all links with the slowest application time in the network domain.

Therefore, the different send cycles of a variety of applications running on an AC d_{ij}, hosting talkers $\boldsymbol{Ta_{ij}}$ creating stream sets $\boldsymbol{S_{ij}}$, do not allow a direct and

immediate link load or throughput measurement and its feedback and control calculation at the network cycle speed. If measured at the network cycle speed, the controller output oscillates with the interference of all different application cycle's data transmissions. Moreover, it would create a considerable CPU load on the PLC to calculate the control loop in every network cycle, which is usually selected within the range of 100 µs to 4 ms, for typical manufacturing automation tasks. The network cycle depends on the applied traffic-shaping method and applications. Furthermore, it would be difficult to collect all the actual values of the throughputs at each link in the network within one network cycle. Therefore, the mean throughput at a link must be measured over a suitable time span. It is evident that this time span T_{mean} minimum length is determined by the slowest application cycle T_{App} sending data over the TSN domain. Under these conditions, it is proposed to calculate as follows:

$$T_{Mean} \geq m \left(\max_{i} \; T_{App\,i}\right) \tag{4-7}$$

where T_{Mean} is the recommended integration time for the calculation of the mean link load or throughput, *m* is an empirical factor that should be selected long enough to smoothen local peaks but short enough to reach sufficient control dynamics. For the simulations of this research task, *m* was chosen as 5, which seemed to be a reasonable choice as a starting point. $T_{App\,i}$ are the application cycles of all the applications in the network domain.

The crucial consequence of these considerations is that the slowest application in the network domain defines the path load measurement integration time and, thereby, the dynamic possibilities of the LDC. The precise consequences of this situation are discussed in Chapter 5 . A distinctive feature in this respect comes from the EST and CQF traffic schedulers, where faster and slower applications could use different traffic classes and thus different EST/CQF windows. This opens up for a load measurement per traffic class, which can be implemented using hardware internal content-aware processors, as they are common in standard TSN switching System-on-a-Chip (SoC) hardware.

Obviously, the load consumption on a path caused by an automation application and measured over an application cycle depends on the frame size and application cycle. This follows directly from Figure 4.8. One might initially assume that smaller loads could simply be switched from one path to the other without even requiring a flow controller to achieve this smoothly. However, the next measurement would also have to wait at least until the settling time to avoid reacting too early with further load shifts. In addition, further exogenous load changes could occur during the transient period, which would negate this approach . Therefore, it is strongly recommended to always include a subsequent proven flow control circuit for path changes of loads, as will be further outlined in Chapter 5 .

4.8 The Influence of Stream Reservation

Stream reservation (SR) can be used in a TSN MAN in combination with different traffic shapers and schedulers to limit the overall network load. This is an effective means to protect the given maximum latency guarantees for data transport. These guarantees are necessary to ensure the functionality of application control loops. SR is defined by the Multiple Reservation Protocol (MRP)/Multiple Stream Reservation Protocol (MSRP) (IEEE 802.1Q, 2022) and Resource Allocation Protocol (RAP) (IEEE 802.1Qdd, 2023).

If SR is applied, it influences the load control properties. SR requires time for the reservation process, which must occur before a stream can flow. This time appears as an additional dead time element in the load-distribution control circuit if the reservation is established dynamically at the control runtime. It is either caused by the distributed reservation protocols MSRP or RAP, or by a central reservation via the Simple Network Management Protocol (SNMP) or the Network Configuration Protocol (NETCONF). The distributed reservation protocols send a Talker Advertise (TA with MSRP) or Talker Announce (TA with RAP) through the network to declare the talker streams. A listener who is interested in a frame achieves a registration and resource reservation of the stream along the path by sending a Listener Ready (LR with MSRP) or Listener Attach (LA with RAP) frame back towards the talker. These paths are typically controlled using a VLAN. These

are rooted at the talker edge bridge and can be found by a path-control mechanism, typically the ISIS-PCR protocol in randomly meshed networks. Alternatively, they can be assigned by network configuration in the case of a fixed topology, which is the case in ring topologies and redundantly coupled rings of MANs. Here, a maximum of two redundant paths exist, which are the two directions into the ring at the ring ports or over the two redundant ring-coupling links. This handshake of stream declaration and registration requires time before the stream can flow on a new path. The MSRP uses MRP as a lower-layer transport protocol for resource reservation. The MRP distributes new information cyclically in fixed cycles, independent of whether there are information changes in the TA or LR attributes. Therefore, with the MRP cycle time selection, a compromise must be found between the fast distribution of new TAs or LRs contents and the limitation of bandwidth consumption by MRP. Typical MRP cycle times are a few hundred milliseconds. This is the time required to transport new reservation-related information from hop to hop. In contrast to MSRP with MRP, RAP uses LRP as a lower-layer transport protocol that directly exchanges attribute content changes between hops at the point in time of their occurrence. Therefore, the reservations via RAP and LRP are to be preferred from the view-point of LDC if the reservation must be changed during runtime. Central resource reservation via SNMP or NETCONF from a CNC also requires time to configure manageable objects. However, this is used in combination with central load control and is not the focus of this study which aims at a distributed concept. There are two strategies for distributed stream reservation for LDCs:

1. **Pre-reservation:** All possible network paths options for a stream to flow are reserved with 100 percent of the stream bandwidth demand. However, only a fraction is used per path, or a different path may be used completely, following the load control calculation result. This has the advantage of highly dynamic path changes, but the disadvantage that bandwidth overbooking must be admitted to use the full network capabilities. As distribution control is never ideal, overbooking must be limited to:

$$B_{MultiPathResMax} = B_{SinglePathResMax} \cdot n \cdot \eta \qquad (4\text{-}8)$$

where $B_{MultiPathResMax}$ is the maximally admissible bandwidth reservation per path for multipath overbooking. $B_{SinglePathResMax}$ is the maximally reservable bandwidth for a single-path network, *n* is the number of available paths, and η is an empirical quality factor of the distribution control with $\eta = f(J);\ 0 < \eta \leq 1$. Parameter *J* is the control quality of the distribution control. It is given by, for example, the integral of the time-weighted absolute error (ITAE) value with $J = \int_0^\infty |e(t) - e(\infty)|\ t\ dt$, where *e* is the control deviation. This is the deviation of the actual value from the setpoint or reference of the control. Pre-reservation is a compulsory precondition if streams are split into several paths instead of completely shifting them between paths. Overbooking must be limited conservatively to ensure that the load deviations do not exceed 100% load per path. To achieve low overbooking combined with high dynamic load control, a compromise could be to take out early stream reservations from load control, that is, to assign a fixed path up to a certain amount of reserved bandwidth. The application of load control must then be initiated for streams that are added after a certain level of reserved bandwidth.

2. **Dynamic reservation:** The shift of a stream completely from a previous path onto a new path involves a new reservation process for the new path just before the shift. This process implies an additional time span, that is, an additional dead time, resulting slower path change. In the case of distributed reservations via MSRP/MRP, this time span consists of

$$T_{ResMSRP} = 2\ n\ T_{Cyc\ MRP}, \tag{4-9}$$

where $T_{ResMSRP}$ is the overall reservation time from the talker to the relevant listener, $T_{Cyc\ MRP}$ is the cycle time in which the MRP attribute changes are forwarded to the next hop, and *n* is the number of hops from the talker to the relevant listener. Factor 2 results from the fact that both the TA towards the listener and the LR toward the talker in the other direction have to make its way along the path.

In the case of distributed reservation via RAP/LRP, this time span consists of:

$$T_{ResRAP} = n\ (T_{TAdv} + T_{LAtt}) \tag{4-10}$$

where T_{ResRAP} is the overall reservation time from the talker to the relevant

listener, T_{TAdv} is the time a TA needs to transition over one hop, T_{LAtt} is the time a LA needs to transition over one hop, and n is the number of hops from the talker to the relevant listener. As the forwarding within the bridge is initiated at the attribute change, which is done by the LRP protocol stack by pure software processing speed in both directions, T_{TAdv} and T_{LAtt} are expected to be approximately the same time.

T_{Res} would appear as an additional dead time element in the flow-controlled path, with a negative influence on the controllability of the data flow.

If stream reservation is to be used, it is recommended to work with pre-reserved resource reservation to fulfill possible high-dynamic requirements. If the dynamic requirements are low, for example, because of only slow applications and the associated long load measurement intervals, the dynamic bandwidth reservation with RAP/LRP can be sufficiently fast.

4.9 Consequences of Network Errors

A possible network error scenario with consequences for the load distribution control is a MAN ring interruption. An interruption of the redundant ring coupling would not be important for considerations in this thesis because it concentrates on ring load distribution. Inter-ring traffic is handled as exogenous traffic from the ring LDC perspective, causing distribution imbalances, as described in Section 4.3. The ring interruption can be caused by either link loss, bridge failure, or failure of a bridged end station within the ring. This would have different consequences depending on the type of traffic.

1. Seamlessly transported CD streams and non-stream traffic would lose transport over one path but would still reach the listeners over the remaining path. Although seamless traffic is not subject to LDC, this loss has consequences on the load distribution for paths in the ring, as one direction from a talker to a listener is cut. The important consequence from the LDC point of view is that this seamless traffic does not cause an increase in load on the remaining paths.

2. Non-seamlessly transported CD streams that used the erroneous link within their path from the talker to the listener need to be shifted to the other alternative ring direction to maintain the stream provision to the listener. This involves either dynamic new bandwidth reservations or the use of pre-reservations by overbooking, as described in Section 4.8. This can result in an overload of paths under certain circumstances. The proposed optimum LDC method in Chapter 6 will also make proposals in Section 6.8 on how to solve this problem.
3. Non-seamlessly transported non-stream CD and non-CD or BE traffic is typically path-controlled by switch-over redundancy protocols such as either RSTP/MSTP in general networks or by dedicated faster protocols for industrial automation, such as MRP (IEC 62439-2, 2021). This traffic is not load-controllable if it is path-controlled by switch-over redundancy protocols, as outlined in Section 4.5. With the switch-over of the path, a new traffic load scenario for the remaining paths evolves. Depending on the traffic shaper and traffic scheduler basis, this can influence higher priority CD streams. With EST, CQF, and ATS, with the possibility of separating traffic by gating windows, this influence can be avoided. With SPQ, it will have a higher or lower influence depending on the configured traffic QoS priorities. An alternative to having this traffic also under load control is to assign it to dedicated VLANs, with managed paths and with disabled FDB learning, that is, under traffic engineering. Then, it can be handled similarly to the non-seamless streams as described in Bullet 2 above.

To summarise the consequences of network errors, a solution is needed on how to react to the loss of a path, especially for non-seamlessly transported CD streams under load control. Otherwise, path overload cannot be excluded. Chapter 6 elaborates on this and proposes solutions in Section 6.8.

4.10 Chapter Summary

In this chapter the various functions and design possibilities for LDC in the specific context of TSN MAN have been analysed. The results show the following key findings:

1. The prevalent topologies for redundant automation networks, a compulsory precondition for load distribution, are ring topology and redundantly coupled rings.
2. The physical topology must be separated into logical paths by assigning them to different VLANs. SPBV is preferred over MSTP for administration-involved path establishment methods. For automated methods, ISIS-PCR with its different redundant tree algorithms path detection types is recommended.
3. A central LDC solution in a CNC has the downside of long reconfiguration calculations and is a single point of failure. This makes the distributed approach better suited for real dynamic load control.
4. A distributed LDC makes only sense if the end station is an influential AC that provides both a reasonable amount of distributable data and sufficient hardware resources. A bridge is an inappropriate location for a sophisticated dynamic load distribution control but could serve as a simpler load distribution switch.
5. Assessments regarding data priority or traffic class assignment strongly depend on the use of traffic shapers and schedulers. These can already be predetermined by the selection of the automation technology.
6. Only a non-seamlessly transported CD is available for LDC, as the reduction in throughput is not an alternative for CD.
7. Distribution control and flow control are considered relevant for this study. Fairness flow reduction of the CD is not permitted for the MAN that is the focus here.
8. Linear dynamic control is the most promising control method because of the typically constant and known ingress data rates and cycles. The linear dynamic control should build the runtime control component and should be based on thoroughly conducted traffic engineering during the network planning and

setup phase to guarantee optimal control results.

9. The load distribution control task is based on the optimisation goal to minimise the maximum load peak along two possible paths within the automation ring. The flow control loop is a sub control loop of the complete distribution control. The data paths from flow control point of view form dead time elements as they delay data transport.
10. The application cycle time plays an important role for the control circuit design. The slowest application in the network domain assigns the minimum integration interval for the measurement of the rolling mean of the loads within the bridges and bridged end stations.
11. If stream reservation is used, it is recommended to work with pre-reserved resource reservations to fulfill demanding dynamic requirements.
12. A strategy for coping with network errors was sketched. These have an impact particularly on non-seamlessly transported CD streams and have to be handled by pre-reservation or dynamic re-reservation. This is further developed in Section 6.8.

The insights gained in this chapter form the basis and the boundary conditions for the design of an optimum LDC in TSN MAN in the next chapters.

Chapter 5 Application of Different TSN Traffic Shapers and Schedulers for Subsequent Data Flow Control

5.1 Introduction

A further step towards finding an optimised control method for load distribution control in TSN MANs is to clarify the influence of the different TSN elements on the network properties and thereby the data flow control properties. As outlined in Section 4.6, data flow control is a sub-control task of the distribution control task. Its purpose is to control the increase and decrease in the data flow on a communication channel.

In this chapter, the model of the automation network path is derived, and the influence of different traffic shapers and traffic schedulers is discussed in more detail. Furthermore, example network simulations are performed.

5.2 The Network as the System under Control

As outlined in Section 4.7, TSN MANs data update rates typically only range between a few microseconds and several hundreds of milliseconds. Applying state-of-the-art slower IT communication network analysis (Gebali, 2015) by collecting all nodes discrete Markov-Chain buffer states at each network cycle, would mean a high network and CPU load for LDC. Therefore, the fast network cycles of an automation network require a more efficient control method. This must be specially tailored to the effectiveness and to the comparably lower available CPU control performance of automation controllers.

To this effect, a central question regarding the load control in communication networks is which characteristics the network has when being modeled as a system under control. The parameter to be controlled is bandwidth consumption which is measured in bit/s and may be normalised to the maximum bandwidth as a percentage of the maximum bandwidth. The data itself experiences only a delay when transferred through the network. This delay is caused by the transfer

through a bridge and by the LAN propagation delay caused by cable inductances and capacities. From a control perspective, these delays represent dead time elements. Therefore, the network, as a system under control, is a series of pure dead time elements that can be combined into one element. The actual dead time depends on the path length from the input to a relevant point in the network that is controlled at a certain point in time. To measure bandwidth usage, a suitable measurement time span must be selected to obtain a stable and non-oscillating measurement value. Within this time span the measurement is carried out as a sliding window measurement or, in other words, a rolling mean measurement. This rolling mean measurement from a control perspective represents a PT1 element in the feedback path. Regarding the controller itself, it appears that a linear controller is at hand as the system is linear under the preconditions above. This discussion and selection of the linear control type are further elaborated in Section 6.3. The core controller is the flow controller which is responsible for increasing or decreasing the data flow along the data paths according to a reference value.

As for CD only the redirection of data from one path to another is acceptable, an increase in data flow on one path always involves a decrease of the same extent on the other path. Flow control is only sensible for streams or non-streams with relevant bandwidth consumption, which is worthwhile to be split into parts to be put onto paths in a continuous transition. Smaller data flows will either be shifted completely from one path to the other without the involvement of flow control or multiples of these single streams will form the minimum resolution steps of the flow control. To determine this, is the task of a mature fully working sophisticated packet controller. However, the ideal design of such is not the focus of this research.

The reference value for the data flow control must be provided by a distribution controller whose output is the result of a comparison between the maximum load values in both directions of the ring for the maximum-reduction control method. It also has the task to weigh this difference between the paths by a factor of 0.5.

This is the maximum decrease which is supplemented by the same increase on the other path, thus achieving equal maxima.

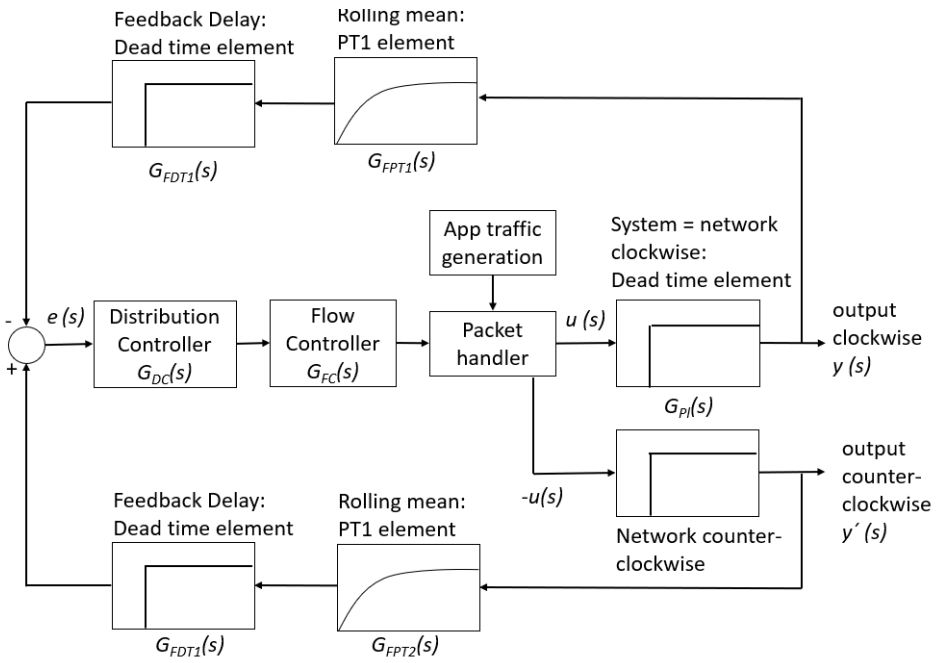


Figure 5.1: Control principle of the distribution control assembly in network rings

As shown in Figure 5.1, the system is a two-part system consisting of the two directions of the ring. Derived from overall distribution control, flow control as a subtask within an influential AC such as a PLC or MC has the need to reduce the load on a given path and shift all or part of it to the alternate path. An influential controller is an AC which transmits sufficient data that can be redirected to contribute to a significant change in the load distribution. As a load decrease in one direction must compulsorily lead to an increase in the same amount in the other direction, it is sufficient to apply flow control on only one path and convert the flow controller output for the other direction. This is achieved by the packet controller on the one hand ensuring the most balanced possible distribution of the applications packets at the start time and on the other hand converting necessary deviations into packet distributions.

This flow control task in switched layer 2 automation TSN networks under the influence of different shapers and different application cycles is the focus of the following sections within this chapter.

As the system under control consists of only dead time elements, plus PT1 elements and dead time elements in the feedback, there is no point in applying the state-space description method. Instead, the classical approach of applying the Laplace-transform in the frequency domain is preferred. There is also no need to work with the z-transformation as the originally time-discrete character of the data values of input and output is not important, as all values are measured as rolling mean values over multiple sampling times. Thus, linear system behaviour is achieved.

It must be stressed that the rolling mean values are smaller than the local and short-time bandwidth maxima which can in principle overload the data transport capability of a path. This can happen, for example, at times when CD bursts occur on the network. However, it is the task of bandwidth reservation such as MSRP (IEEE 802.1Q, 2022) or RAP (IEEE 802.1Qdd, 2023) and ingress limiting functions, as defined by IEEE 802.1Qci (2016), to avoid such local peak overloads. A further practical issue is that the monitoring of the path loads or throughputs and the calculation of the maximum along a certain path is best performed at the distribution controller, where this is needed. The calculation of a sliding or rolling mean value over configurable integration time is best preprocessed by the bridges to keep the network traffic towards the distribution controllers as low as possible.

The sum of the time delays that a data frame experiences while transitioning along a data path from source to destination, from the control perspective, represents a dead time element. Dead time elements in a control plant, and also in the feedback, have the disadvantage that the control system tends easier to oscillations (Normey-Rico & Camacho, 2007). These dead time elements spoil the obvious goal of being able to immediately compensate for possible load changes along the path within the next network cycle. Therefore, a longer control interval should be considered.

To assign the control intervals and later the control parameters, the dead time elements must be analysed.

The line delay or peer delay, together with the bridge latency, forms a dead time element for one hop, that is, the transition over one bridge or bridged end station:

$$T_{DT} = T_{BL} + T_{LPD} \quad (5\text{-}1)$$

With:

T_{DT}: Dead time introduced by one linked bridge.

T_{BL}: Bridge latency time.

T_{LPD}: LAN propagation delay. Typically, it is 5 ns per meter of an Ethernet cable.

The calculation of the bridge latency time depends on the individual traffic shapers and schedulers and is the subject of the next section.

The overall dead time of a given arbitrary network path with m bridges is:

$$T_{DTpath} = \sum_{i=1}^{m} T_{BLi} + \sum_{i=1}^{m+1} T_{LPD} \ , \ m \in \mathbb{N} \quad (5\text{-}2)$$

With:

T_{DTpath}: *Dead time of a given arbitrary network path containing m* bridges.

The dead time function of a system in the time domain is given by:

$$y(t) = u(t - T_{DT}) \quad (5\text{-}3)$$

The dead time system transfer function as Laplace transform is given by:

$$G(s) = \frac{Y(s)}{U(s)} = e^{-sT_{DT}} \quad (5\text{-}4)$$

A dead time element reduces the stability of a controlled plant. It also limits the fastest response time that can be reached by the controller to compensate for the input or disturbance changes. Because of handling and delaying frames in a different way, the different traffic shapers and schedulers of TSN have different bridge delay and path delay characteristics as will be analysed below. Thus, they form different dead time elements.

The physical value of interest for both the input and output of the network as a controlled plant is the data bandwidth which is defined as data per time, measured in bits per second, abbreviated as bit/s or bps as already introduced in 4.6. A sensible measurement interval must be selected to measure the actual bandwidth. Different sources of data traffic in a network often have a variety of traffic forms and send intervals, as outlined in Section 4.7. Therefore, the measurement interval must be sufficiently long to cover all senders sending intervals to achieve a representative and stable measurement value. In addition, for reasons of stability and to filter possible sporadic traffic peaks, experience in control theory recommends applying lowpass filtering, that is, an averaging calculation over time T_{Mean}, of the actual value of the output bandwidth at the bridges and bridged end stations output ports. For example, this could be a cumulative moving average (CMA) implemented by a sliding window algorithm over a number of samples of the output bandwidth μ_M of a bridge port. Another name for this method to be found in the literature is the expression "rolling mean value (RM)," which is preferred in this thesis and was already sketched in 4.6, defined as follows:

$$\mu_{OMean}(t) = \frac{\sum_{i=n}^{n+m} \mu_{Oi}}{m} \; ; n, m \in \mathbb{N} \qquad (5\text{-}5)$$

With:

- μ_{Omean} (t): output bandwidth average (mean) value
- μ_{Oi}: one output bandwidth sample value
- *n:* sample to be the starting point of the sliding window calculation
- *m:* number of samples in the window

In each cyclic RM calculation, an old sample value drops out of the window, and a new value is taken into the window. The pseudo code for building the RM is outlined in Table 5.1. Its detailed structure is provided in Appendix 2.

Table 5.1: Pseudo code to build the RM within the bridges and bridged end stations.

Algorithm: RollMeanApp::Calculate()

This algorithm builds the RM and must be calculated cyclically at least once in each rolling mean integration interval in the bridges and bridged end stations.

```
Create once at instantiation of this method a two-dimensional array m_rm_array [] [] containing a series of structure elements of number RM_WINDOWSIZE of number of received bytes with timestamp, per automation controller ID (ACID), and per application cycle (APPID) ;
Create RM variables: integration time m_inttime, m_currenttime, m_windowstarttime, and m_arraystarttime and initialise them;

Create two-dimensional array p_throughput [] [] to store calculated throughput in an application cycle specific array for throughput progress display and control.
Create index variable m_datapoint for storage of throughputs and initialize it;

Receive and store each port-passing data frame size in number of bytes, timestamp, ACID, and APPID, in m_rm_array via interrupt in a receive method in the background realising a ring buffer;
This receive method supports an index "m_i" which holds the current index of the last storage event into m_rm_array;

Sum up number of bytes for ACID and APPID:
If ((m_currenttime - m_inttime ) >= m_arraystarttime) //window is fully within array
{
  m_windowstarttime = m_currenttime - m_inttime;
  For (i = m_i; m_rm_array [ACID] [APPID].timestamp[i] > m_windowstarttime; --i)
      {
      m_bytes = m_bytes + m_rm_array[ACID] [ APPID].nbytes[i];
      }
}
Else //window suffered a turnover within array or has just started
{
   For (i = m_i; i >= 0; --i)//lower part of the window
       {
       m_bytes = m_bytes + m_rm_array[ACID][ APPID].nbytes[i];
       }
    If (m_rm_array[ACID] [APPID].timestamp[RM_WINDOWSIZE-1] > (Time)(0)) // turnover
      {
      m_windowupperpart = m_currenttime -_
      m_rm_array[ACID][ APPID].timestamp[RM_WINDOWSIZE-1];
      m_windowstarttime = m_rm_array[ ACID][ APPID].timestamp[RM_WINDOWSIZE-1] -
      (m_inttime - m_windowupperpart);
      For ( i = RM_WINDOWSIZE-1; m_rm_array[ACID][ APPID].timestamp[i] >=
         m_windowstarttime; --i)
        {
        m_bytes = m_bytes + m_rm_array[ACID][ APPID].nbytes[i];
         }
       }
}
Calculate throughput and store it in an application cycle specific array for progress display and control:
p_throughput [APPID][m_datapoint]= (m_bytes * 8)/(m_inttime * 10000)
// 1000000 ns per ms divided by 100 is 1 per cent;
m_datapoint++;
```

As a next step, different TSN traffic shaper and schedulers need to be analysed in terms of their influence on the path dead times.

5.3 Applicable TSN Traffic Shapers and Traffic Schedulers

The TSN traffic shapers and schedulers, as introduced in Subsection 2.2.3, will be analysed in terms of the different bridge delays they cause, as these will constitute the dead time elements in the flow control circuit. In the following, the single elements of the timing delay for a hop from bridge to bridge and a complete network path will be analysed for various possible schedulers and shapers.

5.3.1 General Bridge Timing Considerations

TSN offers a variety of traffic shapers and schedulers for the bridge internal MAC forwarding service as defined in IEEE 802.1Q (2022) and IEEE 802.1Qcr (2020). Depending on the forwarding method used, the bridge internal forwarding delay and for some methods also the total path latency is defined.

As outlined in the previous section, the bridge delay of one hop consists of the bridge transit delay or bridge latency T_{BL} and the LAN propagation delay T_{LPD} from the bridge egress port to the next bridge ingress port. The actual transit delay through a bridge depends on several other factors.

A data frame transferred through a bridge experiences several delays in its way from the input port, also called the ingress port, to the output port, also called the egress port.

Figure 5.2 shows the single entities within a bridge that a frame passes during its transition through a bridge if it is forwarded automatically via FDB entries on the data plane, without software involvement, to analyse the frame.

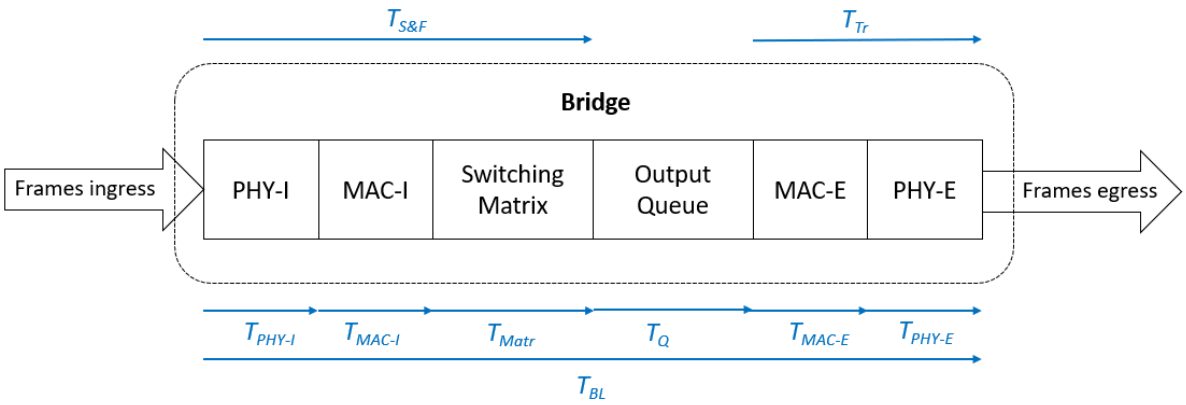


Figure 5.2: Layer 2 bridge internal frame processing entities

The sum of the delays caused by a bridge is called "bridge latency time" or "bridge delay time" T_{BL}. This is defined as follows:

$$T_{BL} = T_{PHY-I} + T_{MAC-I} + T_{Matr} + T_Q + T_{MAC-E} + T_{PHY-E} \quad (5\text{-}6)$$

With:

T_{BL} : Bridge Latency time

T_{PHY-I} : Ingress PHY processing time

T_{MAC-I} : Ingress MAC processing time

T_{Matr} : Switching matrix (also often named "fabric") processing time

T_Q : Time the frame waits in the output queue, which is optional and variable and depends on the amount of data traffic to be sent over the port.

T_{MAC-E} : Egress MAC processing time.

T_{PHY-E} : Egress PHY processing time.

$T_{S\&F}$: Time for frame reception including the time to store the frame and forward it to the output port or output port queue.

T_{Tr} : Time to transmit a frame including times T_{MAC-E} and T_{PHY-E}.

Most of the Layer 2 bridges know both their bridge delay and the delay between end station and bridge or between bridge and bridge, which is called "peer delay" or "line delay," as they usually host a time synchronisation protocol such as PTP (Precision Time Protocol) (IEEE 802.1AS, 2020; IEEE 1588, 2019). This provides a peer delay measurement and bridge delay measurement. Typical bridge delays

without queuing delays range from a few microseconds to a few milliseconds, depending on the bridge technology.

According to IEEE 802.1Q (2022), Annex L.3, and comprising the model from Figure 5.2, the worst-case latency for a frame for a single hop from Bridge to Bridge, can be broken out into the components as listed and assessed for its relevance for load control in Table 5.2.

Table 5.2: Bridge to bridge delay components

delay type	meaning/remark	Relevant for load control?
Input queuing	not relevant here, as there are no input queues in the IEEE 802.1 bridge architecture that constitutes the basis for the bridges underlying this thesis.	no
Interference	depends on the number of non-ring input ports and traffic ingress and is relevant for some of the investigated traffic shapers as queuing delay. Contributes to T_Q.	yes
Frame transmission	is the time it takes to transmit one frame at the transmit rate, which is assumed to be 1 Gbit/s for the networks underlying this article.	yes
LAN propagation	represents the time it takes to send the frame over the LAN to the next bridge depending on the media and distance.	yes
Store-and-forward	consists of all other bridge-internal forwarding elements assuming empty send queues.	yes
Output queuing	is caused by other frames waiting in the output queue to be sent before a frame is due to be sent.	yes

The pure single bridge latency, without traffic depending on the output queuing delay, can be calculated as the store-and-forward delay plus the transmission delay for a frame. The latter depends on the frame size and link speed, assuming no input queues that are usually not common in standard switch ASIC designs:

$$T_{BL} = T_{S\&F} + T_{Tr} \quad (5\text{-}7)$$

The bridge forwarding mode to the output port is assumed to be the store-and-forward mode. The faster cut-through mode is no alternative, as usually more than one input port forwards to the output port. The store-and-forward delay depends on bridge design. A typical value according to IEEE 802.1Qcc (2018), can be assumed to be 700-800 ns.

The transmission delay T_{Tr} is mainly characterised by the actual fame size that needs to be transported through egress MAC and PHY and is thus calculated as follows:

$$T_{Tr} = MaxFrameSize\ [Byte]\ \frac{1}{B}\ 8\ Bit \quad (5\text{-}8)$$

where MaxFrameSize is the maximum SDU size (Service Data Unit = net data load) plus header, usually 42 bytes (IEEE 802.1Q, 2022). *B* is the bandwidth (normally 1 Gbit/s for automation networks), and Bit counts the bits of a byte.

The LAN propagation delay T_{LPD} represents the cable delay from the output port to the next input port. Automation networks are usually set up using copper Ethernet CAT 6 cable which have a specific delay of about 5 ns/m (ANSI/TIA-568.1-D, 2015), that is, a 100 m Ethernet copper cable corresponds to 0.5 µs cable delay. For precise LAN propagation delay assignment, the actual LAN propagation delay from the output port to the next input port can be retrieved from the clock synchronisation peer to peer delay measurement (IEEE 802.1AS, 2020; IEEE 1588, 2019).

Whether the queuing delay T_Q has an influence depends on the forwarding method, that is, the TSN traffic shaping concept. The queuing delay is a variable timing element as it depends on further traffic arriving from other ingress ports and possibly before the frame in question. Such frames will then be in an earlier transmission position in the queue thereby delaying the frame in question. Frame priorities decide the transmission succession and, thus, also the queuing delay.

To evaluate the actual dead times introduced by bridges with various traffic shaping technologies along a data path for Control Data (CD), a dedicated analysis is necessary.

5.3.2 Strict Priority Queuing (SPQ)

The pure strict priority queuing (SPQ) transmission selection is strictly speaking not a modern TSN scheduler, because it does not make advantageous use of any common timing information in coordination with other bridges. Nevertheless, it is still used, especially in combination with higher bandwidth automation networks, for example the 1 Gbit/s and 2.5 Gbit/s PROFINET technologies (IEC 61158-5-10, 2023; IEC 61158-6-10, 2023). A new aspect of this classical technology is that it can be combined with other TSN features. In particular, SPQ can be used together with stream resource reservation which guarantees that frames are serviced to avoid network overload. Thus, a certain determinism is achieved which makes this combination better suited for automation applications compared to the pure SPQ without reservation. SPQ can also be combined with Preemption (IEEE 802.1Qbu, 2015). Preemption allows high-priority frames to interrupt the transmission of lower-priority frames to achieve lower latency for the preemptive traffic class. In this case the high-priority CD, be it I-CD or NI-CD, is assigned the highest SPQ QoS priority, and is the preemptive traffic class. All other lower-priority traffic classes are the preemptable classes. Thereby, minimum latency times are achieved for the preemptive CD. The SPQ can also be applied within an EST window when several traffic classes use a common EST window. In this thesis, both forms, the pure SPQ and the SPQ with Preemption transmission selections are considered.

The SPQ transmission selection for CD needs to assign the highest or second-highest QoS priority to achieve privileged frame handling. This is necessary to achieve the minimum reliable bridge latencies to guarantee the determinism necessary for control tasks. Assuming highest priority for CD and no interfering traffic of the same highest traffic class (In-Class-Interference - ICI) from other controllers along the path, the worst-case situation would be if in each hop along a path, a maximum-sized frame of 1530 bytes (IEEE 802.1Q, 2022) would already

be in the sending process before the CD frame can be forwarded. This frame could not be interrupted with pure SPQ handling capabilities and would delay the forwarding of the CD. To calculate the delay time per hop, Equation (5-7) is expanded by the output port queuing delay T_Q for this disturbing frame:

$$T_{BL} = T_{S\&F} + T_{Tr} + T_Q \tag{5-9}$$

Therefore, for CD with the highest priority with SPQ, the maximum output port queuing delay T_Q is identical to the transmission time of one longest frame transmission time T_{Tr}. If the SPQ is combined with Preemption, the delay T_Q is reduced to the transmission time T_{Tr} of the minimum fragment size, typically 64 bytes (IEEE 802.1Q, 2022). If the CD is assigned only the second-highest priority, this is only acceptable if the requirements for determinism are relaxed, and the highest priority is used only for sporadic network management traffic. For the evaluations in this thesis, the highest QoS priority of 7 was assumed. The overall path dead time with SPQ under the conditions stated above, is determined by the number of hops to be traversed through the network, the delay per hop, and the sum of the LAN propagation delays from the talker to the link of the maximum throughput. It is:

$$T_{DTP\ SPQ} = n_{\max \mu}\,(T_{S\&F} + T_{Tr} + T_Q) + \sum_{i=1}^{n_{\max \mu}} T_{LPD\ i} \tag{5-10}$$

where $T_{DTP\ SPQ}$ is the sum of the dead times of the SPQ path from the controller to the maximum throughput, $n_{\max \mu}$ is the hop count from the controller to the maximum throughput which multiplies the bridge latency from Equation (5-9), and $T_{LPD\ i}$ are the LAN propagation delays between the hops.

Strict Priority Queuing gains attractiveness, especially when it comes to higher-bandwidth systems of 1 Gbit/s and above. The reason is that with a higher bandwidth, the relevance of the maximum frame length transmission time that can block the egress port, that is, T_Q, decreases. Nevertheless, preemption brings a further advantage by minimising the path delay and thus the dead time elements in the control circuit. This makes SPQ an attractive hardware scheduling method in combination with data flow control.

5.3.3 Credit Based Shaper (CBS)

As outlined in the Literature Review in Section 2.2.3, the characteristic property of the CBS is that it expands data bursts to achieve a continuous flow of the stream. Therefore, it is not suitable for industrial automation CD and is, consequently, not considered in this thesis for LDC.

5.3.4 Enhancements for Scheduled Traffic (EST)

The EST or TAS (IEEE 802.1Qbv, 2015) timing calculation is based on the assumption that, with EST, the data can transition through the complete network within a defined gating window. This gating window is synchronised among all nodes in the network domain and is reserved for one or more dedicated traffic classes. A network cycle can be divided into several gating windows assigned to the different traffic classes. The remaining time of the network cycle, which is not consumed by gating windows is usually left to non-CD or best-effort (BE) data traffic with lower timing requirements. Thus, it can also be considered a gating window for this traffic. The necessary length of the gating windows depends on the overall data of the assigned traffic class to be transported per link created by $\boldsymbol{Ta_{ij}}$ stream demand $\boldsymbol{S_{ij}}$ from each end station of $\boldsymbol{D_i}$ at each node $v_i \in \boldsymbol{V}$ along the path. Furthermore, it depends on the maximum length, measured as number of hops, of all possible paths, which is usually limited by the maximum network diameter, and the LAN propagation delays between all hops. The maximum data calculation can be achieved either through network traffic engineering and/or dynamic limitation by stream reservation, with either MSRP (IEEE 802.1Q, 2022) or in the future with RAP (IEEE 802.1Qdd, 2023). With EST, the queuing delay T_Q plays a minor role, because the length of the gating window, and thus the maximum path delay, is fixed. T_Q is thereby part of the gating window and is already considered in the gating window length calculation. Bandwidth reservation by protocol or engineering must secure that the maximum T_Q is not exceeded.

Figure 5.3 shows how the arrival of two frames within a, for example, I-CD EST gating window can be influenced by ICI. Within the gating window T_{GW} a frame 1 arrives after T_1 and frame 2 arrives after T_2 measured from the start of the gating window. Frame 1 has experienced zero queuing delay, only path delays, because of no other ICI traffic on the path, whereas frame 2 has experienced maximum delay owing to maximum ICI.

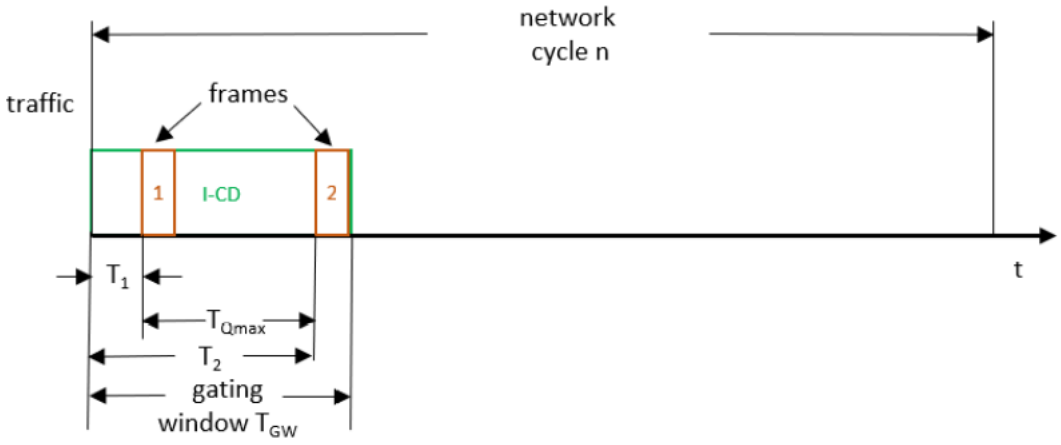


Figure 5.3: Frame arrival during an EST gating window

This maximum delay is identical to the maximum queuing delay T_{Qmax}. T_1 is the sum of all pure bridge delays along the path.

$$T_1 = n_{\max \mu}\,(T_{S\&F} + T_{Tr}) + \sum_{i=1}^{n_{\max \mu}} T_{LPG\,i} \qquad (5\text{-}11)$$

where $n_{\max \mu}$ is the hop count from the controller to the maximum throughput and $T_{LPG\,i}$ are the LAN propagation delays between the hops. $T_{S\&F}$ and T_{Tr} are the store and forward delay and the transmission delay, respectively, as described in Subsection 5.3.1. As shown in Figure 5.3, the exact path delay can have a high tolerance owing to the unforeseen ICI along the path. This tolerance must be considered when optimising the controller parameters. These must always be optimised for the longest possible dead time, as will be seen in the following sections and Chapter 6 . T_1 from Equation (5-11) is thus equal to $T_{DTP\,EST}$ if no ICI exists. The key requirement with EST is to ensure that all cyclic data traffic fits within the gating window. If data are sent unsynchronised from the talker to the

edge bridge, an additional worst-case waiting time of one network cycle time for the next gating window to start must be added.

Especially in connection with synchronised I-CD data injection from end stations, EST achieves minimum overall latencies through the network. Therefore, it is an ideal traffic scheduler from industrial automation load control point of view. Furthermore, the possibility of separating faster application data transport from slower applications by assigning different EST windows offers dedicated load control for groups of applications.

5.3.5 Cyclic Queuing and Forwarding (CQF)

The CQF scheduler (IEEE 802.1Qch, 2019) timing is determined by the number of hops to be traversed through the network and the length of the cycle time plus the LAN propagation delay from the talker to the link of the maximum throughput:

$$T_{DTP\ CQF} = T_{NC}\, n_{\max \mu} + \sum_{i=1}^{n_{\max \mu}} T_{LPG\ i} \tag{5-12}$$

where $T_{DTP\ CQF}$ is the sum of the dead times of the CQF path from the controller to the maximum throughput. T_{NC} is the length of the network cycle, $n_{\max \mu}$ is the hop count from the controller to the maximum throughput, and $T_{LPG\ i}$ are the LAN propagation delays between the hops. The gating window for one traffic class and thus the complete network cycle time can be selected to be smaller than with EST, as with the CQF only one hop must be traversed within the gating window instead of the complete network in the worst case for EST.

As shown in Equation (5-12), the overall CQF path latency is proportional to the number of hops $n_{\max \mu}$ traversed. For each hop a complete network cycle is added, which is a disadvantage, particularly in larger networks with a high network diameter. Although the overall path delay is higher than with SPQ and EST, the influence of ICI on the overall path delay deviation is relatively small. This is only a fraction of a network cycle in relation to several network cycles, one for each hop along the path.

Regarding load distribution control, CQF is a good selection only for smaller networks and a small amount of data per network cycle to keep those small. Otherwise, the path delays and thus, the dead time elements in the control circuit would be relatively high. In turn, this requires greater effort to achieve a robust control circuit.

5.3.6 Asynchronous Traffic Shaper (ATS).

The ATS is the most complex shaper among the various TSN shapers and schedulers and offers a variety of configuration possibilities that would make the timing analysis quite complex. However, the special properties of CD reduce the permissible configuration combinations. First, CD needs to be transported with the highest priority of cyclical frames, in addition to the highest absolute priority of sporadic management frames. Therefore, the ATS IPV with the highest priority must be selected. Second, a burst of CD must also be transported as a burst, that is, it must not be stretched. This means that the committed burst size parameter of the Token Bucket Shaper of ATS must be sufficiently large to guarantee this. The CD data are assigned a reserved stream gate. Unlike the EST, however, the bridges in an ATS domain are not synchronised and unhindered data transport over the entire path is therefore not possible. In the best case, all gates in the bridges along a path open at the same absolute point in time by accident, resulting in a timing similar to EST timing. In the worst case, all waiting times for gate opening when reaching the next hop are maximal. The waiting time per hop is then equivalent to the network cycle time, resulting in a timing similar to the CQF timing. The worst-case overall path dead time with ATS for high-priority CD without ICI is therefore determined by (i) the number of hops to be traversed through the network, (ii) the store and forward delay (no Token Bucket delays for CD), plus (iii) the transmission time and queuing time of one maximum frame, plus (iv) one network cycle per hop, and (v) the sum of the LAN propagation delays from the talker to the link of the maximum throughput:

$$T_{DTP\ ATS} = n_{\max \mu} \left(T_{S\&F} + T_{Tr} + T_Q + T_{NC}\right) + \sum_{i=1}^{n_{\max \mu}} T_{LPG\ i} \quad (5\text{-}13)$$

where $T_{DTP\ ATS}$ is the sum of the dead times of the ATS path for the highest-priority CD. If the maximum ICI is to be considered, a single additional network cycle length must be added.

From Equation (5-13) it follows that the worst-case path delay for CD for an ATS based network is also proportional to the number of hops, as in CQF. A further property of ATS is that the absolute value can only be assigned with a relatively high tolerance owing to the timely asynchrony of the nodes. Therefore, it is also only suited for smaller networks and for smaller amount of data in combination with load distribution control if higher effort for high dead time control circuit design should be avoided. In addition, it creates strongly variable dead time elements that make an LDC difficult to control.

5.3.7 Scheduled Transmission (ST)

The scheduled transmission of data frames refers to a scheduled injection of data frames from the end stations. IEEE 802.1Qcc (2018) provides further definitions of how to support ST. The goal is to provide an orchestrated frame injection from all end stations in such a way that all frames experience a minimum latency without disturbing each other's flow in the network. This demands a centralized schedule calculation using a CNC before network communication can start. Each single change in the network topology or in the communication structure requires a recalculation of the transmission schedules for each end station communication connection, if the claim to optimal distribution is to be maintained. Scheduled transmission must not be mixed up with scheduled traffic as described in Subsection 5.3.4 with the EST scheduler. "Scheduled transmission" means a dedicated send schedule for each frame in the end stations, whereas "scheduled traffic" means reserved gating windows in the bridges for certain traffic classes without differentiating between single frames. Therefore, strictly speaking, scheduled transmission is not a traffic scheduler or shaper method; rather it is a

method of network communication planning. Changes in end station communication demand will lead to a “quasi-dynamic” re-planning and re-configuration into a new static network state rather than to a real dynamic measuring and adjustment. Therefore, ST belongs to the category of traffic engineering rather than to the category of dynamic load distribution which is the focus of this study. It is thereby beyond the scope of this thesis and has not been further investigated.

5.3.8 Discussions and Evaluations

The various traffic shapers and schedulers contribute differently to the path delays. Some are independent from interfering traffic, some provide a rather fixed delay that is relatively independent of the actual hop count, while others change significantly with the hop count.

Bridges using SPQ, SPQ with Preemption, and EST provide rather low path delays also in larger networks. The application of bridges using CQF and ATS can result in significantly high path delay times, and thus, high control dead times, especially in larger networks with a high hop count. Therefore, they are recommended only for smaller network sizes.

ST is a traffic engineering concept rather than a bridge traffic scheduler or shaper, which is beyond the scope of this thesis, as it cannot be applied fully dynamically.

5.4 Identification of the Plant Characteristics

A further problem to be solved with LDC is how to obtain the actual plant properties. The delay dead times are derived from this, which are necessary in order to adapt the control circuits depending on the possible, constantly changing positions of the load maxima.

As outlined in Section 5.2, the control method and design are tightly bound to actual plant properties. In particular, path delays from source to destination are important because they form dead time elements in the control circuit. These parameters are not always at the disposition of a local network and control

designer or are not investigable with reasonable effort. For comparison, the network load control concepts for Internet communication or client/server communication over longer distances lack a detailed knowledge of communication path properties. However, this is easier in MANs, which are typically limited to machines, manufacturing automation cells, and plants. Only newer automation developments, where parts of the automation roles can be virtualized into remote data centers or provided as cloud services, make it increasingly difficult to determine these parameters also in the automation context. Regardless of how the network delays have been assigned, each node containing an AC with load distribution control must maintain an adjacency list in the form of a distance matrix instance ***M***, as described in Section 4.6. This contains the remote nodes and path delays towards them.

Depending on the type of network, various methods for assigning plant properties are available.

5.4.1 Determination at Design Phase

If the load distribution control is to be planned for a MAN with a defined extension such as a field-level ring or a controller-level ring, one possibility is to calculate all possible path delays between all end stations. However, this requires detailed knowledge of:

1. All path lengths, that is, all LAN lengths between end stations and bridges,
2. The store and forward delay $T_{S\&F}$ in each bridge and bridged end station,
3. Transmission time T_{Tr} for a frame. Because this depends on the frame length, it must be calculated with the worst-case packet length or with the longest CD packet length in that network.
4. The traffic shaper or traffic scheduler technology used in the bridges and bridged end stations. These have an impact on the path delay calculation as outlined in Section 5.3.

For the last point regarding traffic shaper and scheduler technology, it is also important whether the technology is homogeneous along all paths or whether there are technology transitions in between. If it is not homogenous, this can

imply additional frame transport waiting times. Such would occur if, for example, frames transition from SPQ to EST, which would require waiting time for the next gating window.

With networks of smaller spatial extension, such as in a machine, the actual LAN propagation delay can be neglected and its continuous measurement during runtime can be avoided. This is possible for EST and CQF. T_{DTP} can be calculated as the constant single dead time per hop, and the number of hops is known. For example, delay with EST can be calculated in good approximation as the relative distance of the throughput maximum in relation to the complete ring length in hop counts, provided that deviations caused by LAN propagation delay differences can be neglected:

$$T_{DTP\ EST} = \frac{T_{GW}}{n_{\max ring}} n_{\max \mu} \tag{5-14}$$

where $T_{DTP\ EST}$ is the overall EST path dead time from the controller to the maximum throughput, T_{GW} is the length of the gating window, $n_{\max ring}$ is the maximum hop count of the ring, and $n_{\max \mu}$ is the hop count from the controller to maximum throughput. The same path delay calculation can be applied figuratively to the CQF. This is, of course, an average value, and there is still the problem with the early insertion of ICI on the path as described in Subsection 5.3.4. Therefore, for controller optimisation of the Load Distribution Control, it is more secure and easier to always take the full gating window size as the one-way path dead time. However, it must be accepted in this case that the convergence time for the load distribution improvements worsens as outlined in more detail below.

As can be seen from the necessary steps listed at the beginning of this subsection, setting up a database with all the exact path delays between all end stations can be quite a large and tedious task. Even with the help of dedicated automated calculation programs, there is still a task left to provide all of these inputs. Another possibility is to use runtime methods to determine the path delays, thus avoiding necessary input from the system administrator. These methods are crucially influenced by the availability of time synchronisation for nodes.

5.4.2 Runtime Method for Unsynchronised Networks

To select an appropriate method for constantly running plant timing properties identification, it is important to determine whether the network and the end stations support a time synchronisation method such as PTP (IEEE 1588, 2019), or gPTP (IEEE 802.1AS, 2020), or whether they are unsynchronised. In an unsynchronised network it is not possible to state the time difference between any two nodes in the network. However, one method to still obtain the data about properties and dimensions of a network and then to draw conclusions about the behaviour over time is to use dedicated test messages.

Test messages are sent from talkers and returned by potential listeners back to the talkers. This system has already been used in a rather simple form of a measured round-trip time (RTT) by TCP congestion control. Further work built on this, such as that provided by Katabi et al. (2002), who investigated high bandwidth delay product network congestion control, as outlined in Subsection 2.4.2. For optimised load distribution control within MANs, test messages are also an option to be used as a basis, but they demand a much more differentiated view. To gain knowledge about the path delay from a given talker to a listener of interest, without any knowledge of the properties of the path elements, the most obvious method is to measure the time difference between the transmission and reception of the reply. However, this presupposes that the talker supports at least a free running own clock or timer with sufficient precision. To be allowed to derive the path delay in one direction as the half from the RTT value it is important though that the path is identical in both directions, that is, that it is "reverse path congruent," and that they are tolerably symmetrical in both directions from a path delay point of view.

$$
\begin{aligned}
T_{TL} \approx \frac{T_{RTT}}{2} \; ; \; for \;\; P_{TL} = \; P_{LT} \; ; \\
P_{TL} = \{ta, v_1, \dots, v_n, li\}; \\
P_{LT} = \{li, v_n, \dots, v_1, ta\}; \\
n \in \mathbb{N}; \;\; P_{TL}, P_{LT} \;\; \in G(V, E);
\end{aligned}
\tag{5-15}
$$

T_{TL} is the path delay from a talker *ta* to listener *li* according to the annotation in Section 4.6. T_{RTT} is the round-trip time from the talker to the listener and back, P_{TL} the path from the talker to the listener, P_{LT} the path from the listener to the talker, and *v* are the bridges or bridged end stations along a path. T_{TL} can never be expected to be exactly half of T_{RTT}. Some bridge delays can depend on different traffic interferences from other than path-ports or from other applications inserting traffic along the paths. The influence of interfering traffic is particularly noticeable in principle with SPQ and ATS. However, resource reservation is an adequate means of limiting this influence in tolerable borders. Traffic interferences have less impact on EST and CQF because a frame cannot experience a greater delay here than when it is at the end of the gating window (with EST) or in the last network cycle (with CQF). The RTT method is appropriate if not all network paths are known or only partly known. It is important that a certain path symmetry can be assumed. Particularly in the context of SPQ or ATS schedulers, with the increased influence of traffic interference on the resulting delay parameters, it is recommended to work with mean values over a period of test intervals. Thus, the influence of short interference traffic peaks is dampened. The mean value integration interval selection is a compromise between the actuality of the values, the thereby created additional load in the network, and the CPU time consumption of the nodes by updating the test packets. Further insecurity with this estimation is introduced by the time consumed by the listener to receive the test frame and send back the response frame. This frame processing can depend on the runtime of other CPU tasks and can therefore also be subject to variation. This is also an argument for mean value use. These insecurities must be compensated by applying an adequate security margin to T_{TL}. A dedicated extension of the round-trip packet for the use of LDCs in MANs is the installation of a hop counter in the packet as discussed in the previous subsection. The test messages are modified by the nodes along the path by updating the counters. Thus, the distance between the listener load distribution controller is known. This information is useful for deciding whether streams to the listener influence path links. However, this method can only be applied within smaller automation setups

which include tailored nodes for LDC, as such a feature requires a dedicated control protocol and thus a load control awareness of each node in the TSN MAN.

5.4.3 Runtime Methods for Synchronised Networks

In a synchronised network, all end stations, bridges, and bridged end stations share a common time. Because of the time synchronisation protocols necessary to operate on each bridge, they know both their own bridge latency time and the path delay to their neighbouring bridges. This information is accessible to a CNC via, for example, SNMP or Netconf/Yang from the peer delay values in the management information database (MIB) of the time synchronisation protocols. From there, it can be made accessible to each AC in the automation network. Depending on the different path delay calculations of the different traffic shapers and schedulers, ACs can thus calculate the individual path delays to each connected listener in the network domain.

Another possibility with synchronised networks is that each listener would announce the individual path delays from certain talkers to itself. Resource reservation protocols such as MSRP and RAP provide parameter values named “accumulated latency” in their talker announce frames. When a talker announces its streams into the network on the control plane, this accumulated latency field is updated by each traversed bridge with its latency contribution. The original idea of this parameter is that it serves as a decision basis for a listener whether the delay of a stream along a path is acceptable and the listener can subscribe to that stream. In this case, it sends a Listener Attach frame back to the talker. Then, each involved bridge along the path back reserves bandwidth resources. This mechanism can be extended for load distribution control in the following manner:

1. Talker Advertise (MSRP), or Talker Announce (RAP) frames obtain an application-specific TLV with a hop count parameter in addition to the accumulated latency parameter which is incremented by each bridge along a path.
2. The MSRP Listener Ready or RAP Listener Attach frames, both of which have the same core functionality, are extended by an application-specific TLV

containing the accumulated latency and hop count from the talker to this listener.

3. Thus, the path latency and hop count to the listener are communicated to the stream origin talker by the Listener Ready/Listener Attach frames.
4. At the talker end station, a database for all addressed listeners' distance and latency must be supported.
5. As Talker Advertise (MSRP), Talker Announce (RAP), Listener Ready, and Listener Attach frames are repeated cyclically, the latencies are always up-to-date.

It must be stressed though that also here the calculated path delays are only an estimate, as the actual path delays are never constant. As already outlined above in connection with the unsynchronised network and in Section 5.3, the calculations for the path delays of the different traffic shapers and schedulers are all influenced to their individual extent by ICI queuing delays. This is also an issue when updating the accumulated latency parameters. However, this insecurity associated with accumulated latency is not addressed in the protocol standards for MSRP (IEEE 802.1Q, 2022) and RAP (IEEE 802.1Qdd, 2023). The difficulty of a bridge is determining whether to add its own latency contribution with or without a certain amount of ICI for the accumulated latency path calculation. Figure 5.4 shows the problem using an example of the EST scheduler. It shows the course of the actual accumulated latency (black graph) build-up of a stream s_{11}^1 originating from talker ta_{11}^1 of AC device d_{11} connected to bridge node v_1. At egress at v_1 the stream experiences a longer delay caused by ICI s_{21}^1 which arrived earlier at the egress queue than s_{11}^1. This delay can be increased if ta_{11}^1 would send outside the synchronised gating window, as shown in the graph. Theoretically, all possible ICI could already be inserted and reserved at the edge bridge (v_1 for s_{11}^1).

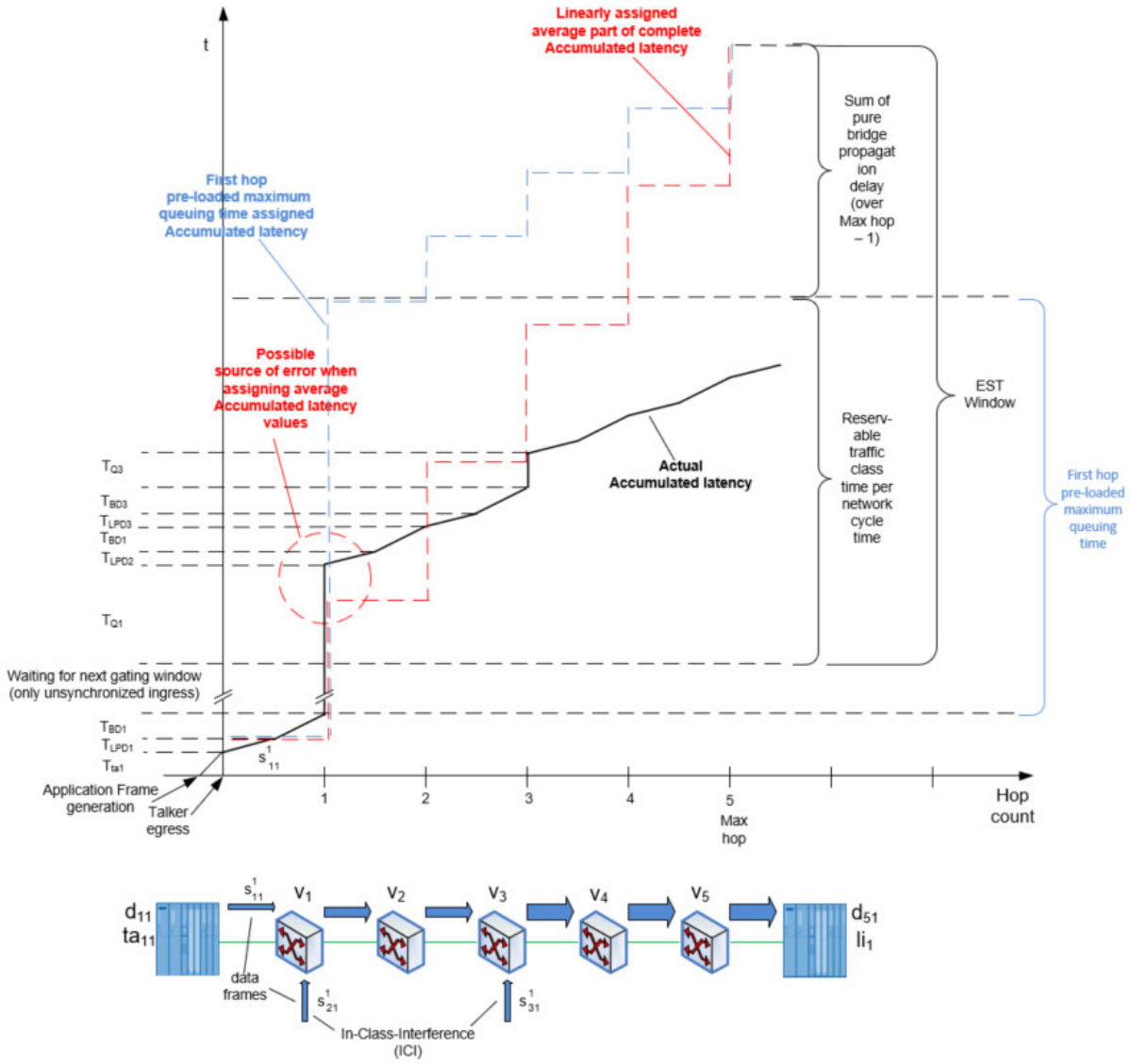


Figure 5.4: Accumulated Latency along a path.

The red dashed line shows the build-up of the accumulated latency parameter when the maximum ICI, and thus the maximum T_Q, is distributed among the number of possible nodes on the longest path.

$$T_{Q\ Node\ EST} = \frac{T_{Qmax}}{n_{max}} \tag{5-16}$$

$$T_{AL\ Node\ EST} = T_{S\&F} + T_{Tr} + T_{QNode\ EST} + T_{LPD} \tag{5-17}$$

$T_{Q\ Node\ EST}$ is the average hop queuing delay of EST and $T_{AL\ Node\ EST}$ is the increment in the accumulated latency parameter per hop for EST. $T_{S\&F}$ is the store and forward delay, T_{Tr} the transmission delay, and T_{LPD} the known LAN propagation delay as outlined in Subsection 5.3.1. In this case, the actual

accumulated latency would be higher than the parameter value when egressing at v_1 (red dashed circle). This case must not happen, because a listener connected to v_1 would then base its subscription on a wrong lower guaranteed latency which can spoil application control circuits. In addition, from LDC point of view, a control circuit optimisation on lower delays than actually present would mean an output overshot which is especially bad with load distribution control as explained in more detail in the following sections. A better approach is to add all possible queuing delay at the talker edge bridge, that is, at the first hop in the network.

$$T_{AL\ TEdge\ EST} = T_{S\&F} + T_{Tr} + T_{Qmax} + T_{LPD} \quad (5\text{-}18)$$

$$T_{AL\ Node\ EST} = T_{S\&F} + T_{Tr} + T_{LPD} \quad (5\text{-}19)$$

$T_{AL\ TEdge\ EST}$ is the accumulated latency increment for the talker edge bridge. The blue dashed line in Figure 5.4 shows the course of this accumulated latency assignment method. With this method, the error during egress at v_1 is avoided. A higher calculated accumulated latency value than the real accumulated latency is not problematic. The controller parameter optimisation for longer dead times would only be expressed by a slower convergence time but not by an overshot of the output, as will be shown in more detail in the following sections. Another more accurate procedure is to estimate the bridge local queuing delay per egress port by means of stream reservations made by the MSRP or RAP. The bridge then calculates the worst-case ICI for each ingress/egress port combination. This is still a worst-case calculation because the actual delay depends on the absolute position of the frame in the queue, which cannot be predicted. It depends on the time of transmission in the various end stations in the network, which is subject to device-local decisions.

In summary, it can be stated that all accumulated latency update methods will result in higher or lower delay time variations depending on the gating window length and the path delay without interference. The question is for which value the control circuit parameters must be optimised. The best mitigation for this problem is to design the controller parameters, for example, in the case of a PID

controller, for the longest possible delay to avoid an overshot of the data flow output as mentioned above.

5.5 Control Method and Structure

Figure 5.5 depicts the control structure of a single network path with a plant consisting of all network hops latency times and LAN propagation delays appearing as dead time elements.

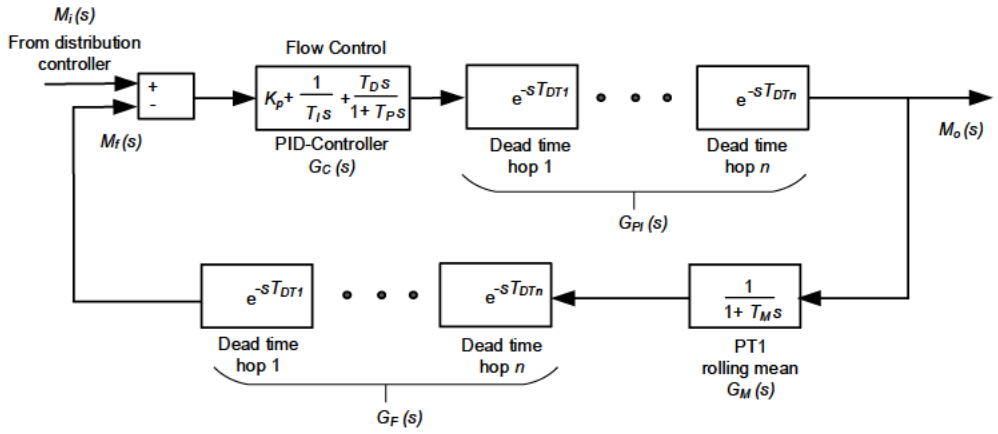


Figure 5.5: Network path flow control structure

The feedback path contains a PT1 element caused by the rolling mean calculation of the feedback. It also contains all hop dead time elements caused by the transition time of the feedback data from the relevant link back to the flow controller in the AC. The relevant position of the link with the current maximum throughput, and thereby the number of hops between the controller and link, is determined by the maximum throughput of all links along the path from a talker to the last listener. The controller is designed as a real PID controller, that is, it contains a parasitic PT1 element. $M_i(s)$, $M_o(s)$ and $M_f(s)$ are the Laplace transforms of the throughput $\mu(t)$. These are the setpoint or input throughput $\mu_i(t)$, output throughput $\mu_o(t)$, and feedback throughput $\mu_f(t)$. K_P is the proportional gain, T_I the integral time, and T_D is the derivative time of the PID controller. T_P in the PT1 element in the denominator of the derivative part of the PID controller represents the real PID behavior that contains parasitic filters. To assign the PID controller parameters K_P, T_I, and T_D, tuning according to Ziegler-

Nichols or Chien-Hrones-Reswick (Normey-Rico & Camacho, 2007) is applied to a plant involving feedback and simulating the step response at the open loop at $M_f(s)$. Experience shows that further empirical fine-tuning of these parameters in a "try-and-error" fashion can improve the Ziegler-Nichols or Chien-Hrones-Reswick parameter assignments. The pseudo code for the core of the PID Controller is outlined in Table 5.3. Its detailed structure is provided in Appendix 2.

Table 5.3: Pseudo code of algorithm for the PID Controller.

Algorithm: PIDCtrlApp::Calculate ()

This algorithm implements the core of the PID Controller. It can be used for both the data flow controller and the distribution controller in an AC.

```
Create variables:
m_ref;        // reference input r(t).
m_kp;         //Proportional factor
m_ki;         //Integral factor
m_kd;         //Differential factor
m_out;        //Output y(t)
m_int;        //integral sum up
m_lastint;    //last integral sum up
m_lasttime;   //point in time of last calculation
m_intstep;    //integration time step in ns, that is calculation cycle for PID controller
m_lastref;    //reference at last calculation

Load m_ref with the result of the minmax comparison of Table 4.4.

If |m_ref| > threshold
{
  Build integral part:
  m_int = m_lastint + m_ki * m_ref * (current_time - m_lasttime);

  Add proportional and differential part:
  m_out = m_kp * m_ref + m_int + ((m_ref - m_lastref)/(current_time - m_lasttime) * m_kd);

  Store current results for next summation cycle:
  m_lastref = m_ref;
  m_lasttime = current_time;
  m_lastint = m_int;
}
Schedule next calculation cycle after integration interval m_intstep;
```

The transfer function of the plant $G_{Pl}(s)$, that is, the network path in the frequency domain, is given by:

$$G_{Pl}(s) = e^{-T_{DTP}\, s}, \tag{5-20}$$

with:

$$T_{DTP} = \sum_{i=1}^{m} T_{DTi}\,, \tag{5-21}$$

where $m \in \mathbb{N}$ is the number of hops from the controller to the link with the current maximum of the throughput along the path. T_{DTP} is the sum of the dead times of these hops, consisting of the bridge latencies and LAN propagation delays. The transfer function of the closed loop G_{CL} is then

$$\begin{aligned} G_{CL}(s) &= \frac{M_o(s)}{M_i(s)} \\ &= \frac{G_C(s)G_{Pl}(s)}{1 + G_C(s)G_{Pl}(s)\, G_M(s)G_F(s)} \\ &= \frac{(K_P + \frac{1}{T_I s} + \frac{T_D s}{1 + T_p s})e^{-T_{DTP}\, s}}{1 + (K_P + \frac{1}{T_I s} + \frac{T_D s}{1 + T_p s})e^{-T_{DTP}\, s} \frac{1}{1 + T_M s}\, e^{-T_{DTF}\, s}}, \end{aligned} \tag{5-22}$$

where the product $G_C(s)G_{Pl}(s)\, G_M(s)G_F(s)$ in the denominator is the transfer function of the open loop $G_0(s)$.

One important goal of automation data control is that no or only a minimum of data frames may be lost to avoid bumps in the controlled process. Therefore, an overshot of $M_o(s)$ over the reference level $M_i(s)$ must be avoided, because the operating point could be near the maximum bandwidth. An overshot would then mean a congestion loss. As the plant consists of only dead time elements, this limitation is equivalent to the requirement for proportional gain: $K_P \leq 1$. Another reason for this limitation is the practical aspect: an overshot would mean an oscillation of the load between two paths, which would only create unnecessary disturbances. However, the price of the avoidance of overshot means slower dynamic performance.

Generally, dead time elements increase the difficulty of controlling the loop and promote its tendency toward instability. However, because of the PT1 dampening effect of the rolling mean calculation in the feedback, the instability of the control loop can be counteracted if the sum of the dead time elements is small compared

to the T_{Mean} of the rolling mean calculation. T_{Mean} increases with the longest application cycle, and is thereby determined by the slowest application, as stated in Equation (4-7). The sum of dead times depends on the selection of the traffic shaping technology, the number of hops between the controller and the current throughput maximum, and the LAN propagation delays of the links between the hops. With certain traffic shaping methods, bridge delays can be assumed to be nearly constant, whereas others imply variable bridge delays and thereby variable dead time elements in the control circuit. A nearly constant bridge latency and thereby constant dead time element as it is given with, for example, EST traffic shaping, has the advantage that T_{DTP} does not need to be measured and transferred to the controller continuously. T_{DTP} can be calculated instead if a constant single dead time per hop and the number of hops are known. However, if the dead time needs to be measured as described in Section 5.4, it is recommended to perform this continuously in a parallel process to the actual throughput control to obtain instant dead time values for the load control.

The general control structure in Figure 5.5 provides separate overall dead times for the data path and feedback path. This is because the values are not always identical. The paths to be followed in the two directions to and from the relevant link are not necessarily the same and could have different delays owing to the influence of the interfering traffic. The local maximum of the throughput $\max_{i,j \in \boldsymbol{V}} \mu_{ij}$ can be at different locations in the network domain at each distribution control loop sample time, resulting in different path characteristics. These in turn demand different controller parameters for the flow control, if optimal flow control is to be achieved. This means that the controlling instance located within an AC must provide and use dedicated plant models for each possible location of $\max_{i,j \in \boldsymbol{V}} \mu_{ij}$.

A common way to identify the influence of dead time, and thereby the difficulty in controlling the control loop, is the use of a normalized dead time related to the time constants of the delaying elements, that is, the PT1 element in this case, such that (Normey-Rico & Camacho, 2007):

$$\tau = \frac{T_{DT}}{T_{DT} + T_n} \tag{5-23}$$

where τ is the normalised dead time, with $0 \leq \tau \leq 1$, T_{DT} is the real dead time of the plant and T_n is the delay time constant of the plant. If τ is near 1, usually $\geq \frac{2}{3}$ (Normey-Rico & Camacho, 2007), a system, as a rule of thumb, is said to be dead time dominant; otherwise, it is said to be lag dominant. The PID controller in the control structure in Figure 5.5 is sufficient, if dead times are relatively small in comparison to the PT1 rolling mean element, that is, the system is rather lag dominant. If the system is rather dead time dominant or demands enhanced dynamics, the PID controller should be replaced using a predictive controller that is either a Smith Predictor (De Cicco et al., 2011; Mascolo, 2000) or a Model Predictive Controller (Normey-Rico & Camacho, 2007).

In summary, it must be stated that the different traffic shaper and schedulers introduce different dead time elements into the flow control circuit influencing the control characteristics. Furthermore, the network cycle and application cycles in the network domain play a crucial role, as they affect the possible control performance by assigning the possible rolling mean time constant of the throughput, and thereby the relation of dead times to delay time constants.

5.6 Network Flow Control Simulation and Results

As described in the previous section, the characteristics of the plant model, formed by a network path in this case, are given by the typical internal bridge delays, line delays, and the way to build the output value which is the mean of the output bandwidth.

This model is introduced into the mathematical simulation tool MATLAB with its control engineering extension, Simulink. The mathematical simulation of the network path provides detailed data and knowledge of the dynamic characteristics of the network path, both as an uncontrolled dynamic system and closed-loop controlled dynamic system. Such dynamic characteristics include system step response behavior, control instability areas, input frequency influences and dynamic reaction speed.

One result of the data traffic analysis in Section 5.3 is that the actual dead time for networks with a similar number of hops and throughput depends on the different traffic-shaping methods. To compare their influences, a sample network model was simulated as shown in Figure 5.6.

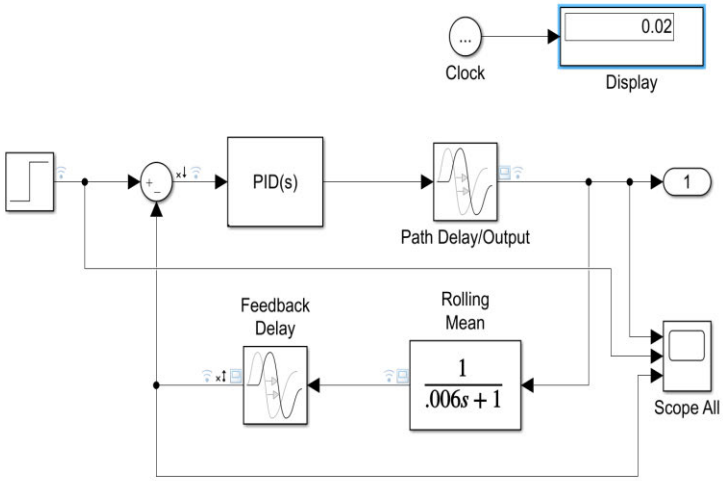


Figure 5.6: Network control simulation model

The parameters for the path delay and feedback delay are calculated as follows and summarised at the end of this section in Table 5.4. The simulation parameters are listed in Table 5.5. A network of 25 hops from the controller to the link with the current maximum throughput $\mu_{ij\,max}$ is assumed, which is half the typical maximum ring diameter of 50 hops (IEC 61158-5-10, 2023; IEC 62439-2, 2021). The average cable length between the hops is assumed with 100 m Ethernet CAT6 cable which have a typical propagation delay of about $0.5\ \mu s$ (ANSI/TIA-568.1-D, 2015). Thereby $T_{LPD} = 24\ x\ 0{,}5\ \mu s = 12\ \mu s$, under the assumption that the controller is near the first bridge with insignificant LAN propagation delay. A maximum data amount of 100 streams with a maximum of 200 bytes net SDU data load plus a 42 Byte Ethernet header is assumed. For one stream frame, this leads to a transmission delay according to Equation (5-8):

$$T_{Tr} = 242\ Byte \frac{\left(8 \frac{Bit}{Byte}\right) 10^{-9} s}{Bit} = 1.936\ \mu s \tag{5-24}$$

Therefore, to a Bridge Latency time according to Equation (5-7) of,

$$T_{BL} = T_{S\&F} + T_{Tr} = 0.800\ \mu s + 1.936\ \mu s \approx 2.75\ \mu s \tag{5-25}$$

For **SPQ** without considerable ICI, the dead time required to shift one frame from the talker to the link with the maximum throughput is according to Equation (5-10):

$$\begin{aligned} T_{DTP\ SPQ} &= n_{\max \mu}\left(T_{S\&F} + T_{Tr} + T_Q\right) + \sum_{i=1}^{n_{\max \mu}} T_{LPG\ i} \\ &= 24\left(0.8\ \mu s + 1.936\ \mu s + 1530\ Byte \frac{\left(8\frac{Bit}{Byte}\right)10^{-9}s}{Bit}\right) \\ &+12\ \mu s \\ &= 371.24\ \mu s \\ &\approx 370\ \mu s \end{aligned} \tag{5-26}$$

The path delay with Preemption, where only 64 Byte instead of 1530 Byte are to be calculated for T_Q, would result in 89,9 µs ≈ 90 µs path delay. These low path delay values for SPQ are a result of the assumption that no other interfering ICI enters the path that would raise T_Q. If the worst case is assumed for this example, the rest of the maximum load enters the ring at a ring interconnection to a coupled ring in between, and this data is in front of the control data, one further $T_{Q\ ICI}$ of:

$$T_{Q\ ICI} = 99 \times 1.936\ \mu s \approx 200\ \mu s \tag{5-27}$$

would have to be added, resulting in a dead time of approximately 570 µs for SPQ with ICI.

For **EST** with ICI, calculation of required gating window length is necessary. To shift the maximum data of 24200 Bytes through the network along the path, one T_{BL} of 195 µs (as reception and forwarding of bytes from bridge to bridge occur nearly simultaneously) plus the complete LAN propagation delay of $T_{LPD} = 12\ \mu s$ is to be calculated. This results in a minimum gating window time T_{GW} of 207 µs. This time also represents the worst-case delay for the I-CD data if the talker transmits synchronised with the network gating windows. For unsynchronised talkers for NI-

CD, one network cycle of worst-case waiting time must be added, which would then result in a delay of 1207 µs assuming a network cycle time of 1 ms. For EST without ICI the delay would be according to Equation (5-11):

$$\begin{aligned} T_{DTP\ EST} &= n_{\max \mu}(T_{S\&F} + T_{Tr}) + \sum_{i=1}^{n_{\max \mu}} T_{LPG\ i} \\ &= 25(0.8\ \mu s + 1.936\ \mu s) + 12\ \mu s \\ &= 80.4\ \mu s \\ &\approx 80\ \mu s \end{aligned} \tag{5-28}$$

For **CQF**, one network cycle time is required to transport the data over one hop. According to Equation (5-25), this needs to be at least $T_{BL} \approx 195$ µs for all 100 streams of this example, assuming that this data is the only traffic class to be transported within the network cycle. The LAN propagation delay must be added to reach the next hop. The overall delay from the controller to the link with the current maximum throughput $\mu_{ij\ max}$ is then according to Equation (5-12):

$$\begin{aligned} T_{DTP\ CQF} &= n_{\max \mu} T_{NC} + \sum_{i=1}^{m} T_{LPG\ i} \\ &= 25 \times 195\ \mu s + 12\ \mu s \\ &= 4{,}887\ \mu s \\ &\approx 4{,}890\ \mu s \end{aligned} \tag{5-29}$$

As with SPQ, if the maximum ICI is considered, the dead time would have to be increased by a further 200 μs.

For **ATS,** the same network cycle time as that of CQF is assumed to be the only CD traffic class to be transported. According to Equation (5-13), the worst-case path delay for the network path under simulation must be calculated as follows:

$$T_{DTP\ ATS} = n_{\max \mu}\left(T_{S\&F} + T_{Tr} + T_Q + T_{NC}\right) + \sum_{i=1}^{n_{\max \mu}} T_{LPG\ i}$$

$$= 25(0.8\ \mu s + 1.936\ \mu s + 1530\ Byte\frac{\left(8\frac{Bit}{Byte}\right)10^{-9}s}{Bit} + 195) + 12\ \mu s \quad (5\text{-}30)$$

$$= 5{,}141.4\ \mu s$$

$$\approx 5{,}140\ \mu s$$

As with SPQ and CQF, if the maximum ICI is considered, the dead time would have to be increased by 200 μs.

Bandwidth Reservation reconfiguration dead times are not considered in the simulations for two reasons. First, the dynamic changes of reservation have practically only relevance in networks, including slow applications; otherwise, their part of the dead time would be a multiple of the dead time caused by traffic shapers and schedulers. Second, it would only add dead time of the same amount for all the investigated shapers and schedulers thus distorting the view of the actual results.

Table 5.4 summarises the path dead time results for the different traffic shapers for the simulated network.

Table 5.4: Path dead times for the different traffic shapers and schedulers

Traffic shaper and traffic type	Worst case Path dead time (μs)
SPQ without ICI	370
SPQ with Preemption and without ICI	90
SPQ with maximum ICI	570
EST without ICI	80
EST with maximum ICI	280
CQF without ICI	4,890
CQF with maximum ICI	5,090
ATS without ICI	5,140
ATS with maximum ICI	5,340

As outlined in Section 4.7 and 5.5, the influence of these dead times is only dominant in networks that are not informed by slow applications, forcing a multiple of these times as the integration time for the rolling mean calculation.

To visualise the influence of the different dead times on LDC, a high-performance application with an application cycle of only 2 *ms* was simulated. The integration time for the rolling mean calculation of the actual value feedback was selected to be five times the application cycle of 2 ms, that is, *m* of Equation (4-7) is 5, which is equivalent to a time constant of approximately 6 *ms* for the PT1 time constant T_{Mean}. The PID controller is optimised for minimum overshot rather than for fast setpoint approximation for the reasons mentioned in Subsection 5.4.3. The dynamic behaviour is analysed through the reference step response. As the control circuit contains dead times that imply non-rational elements in the transfer function, a stability analysis via poles and zeros of the closed-loop or

open-loop system is not at hand. Instead, the Nyquist criteria for the open loop delivers evidence as to the stability and robustness of the closed-loop flow control, that is if the magnitude of the transfer function of the open loop $|G_0(s)| = |G_C(s)G_{Pl}(s)\ G_M(s)G_F(s)| < 1$ (compare to Equation (5-22)), at Im $(G_0(s)) = 0$, the closed loop is stable. The factor for the gain at Im $(G_0(s)) = 0$ to reach $|G_0(s)| = 1$, that is, the gain margin, should not be smaller than 2 ≙ 6 *dB* for a robust control design stability reserve. The second stability criterion is the phase margin, which represents the angle of $G_0(s)$ with the negative real axis at the point of intersection with the unitary circle $|G_0(s)| = 1$. For a robust control design, the phase margin should be ≥ 45°. A Padé approximation of order 16 has been applied for the linearisation of the dead time elements.

Figure 5.7 to Figure 5.9 show the simulation results for the step response and Nyquist diagrams for a representative selection of three networks and traffic situations featuring EST, SPQ and ATS. The simulation parameters are listed in Table 5.5.

Table 5.5: Simulation parameters

Traffic shaper and traffic type	**Worst case Path dead time (*µ*s)**	**Simulation time (ms)**	**PID K_P**	**PID T_I (ms)**	**PID T_D (ms)**
EST without ICI	80	20	0.75	7.7	0
SPQ with maximum ICI	570	20	0.95	6.5	0
ATS with maximum ICI	5340	100	0.38	22	0

Strictly speaking, the use of ATS in combination with a fast application cycle of 2 ms makes little sense from the application control point of view. This is because the data transport for the setpoint and the actual value would be longer than the overall available time to calculate an application control algorithm, including the data transport times. Nevertheless, this is investigated here for reasons of flow control behaviour analysis.

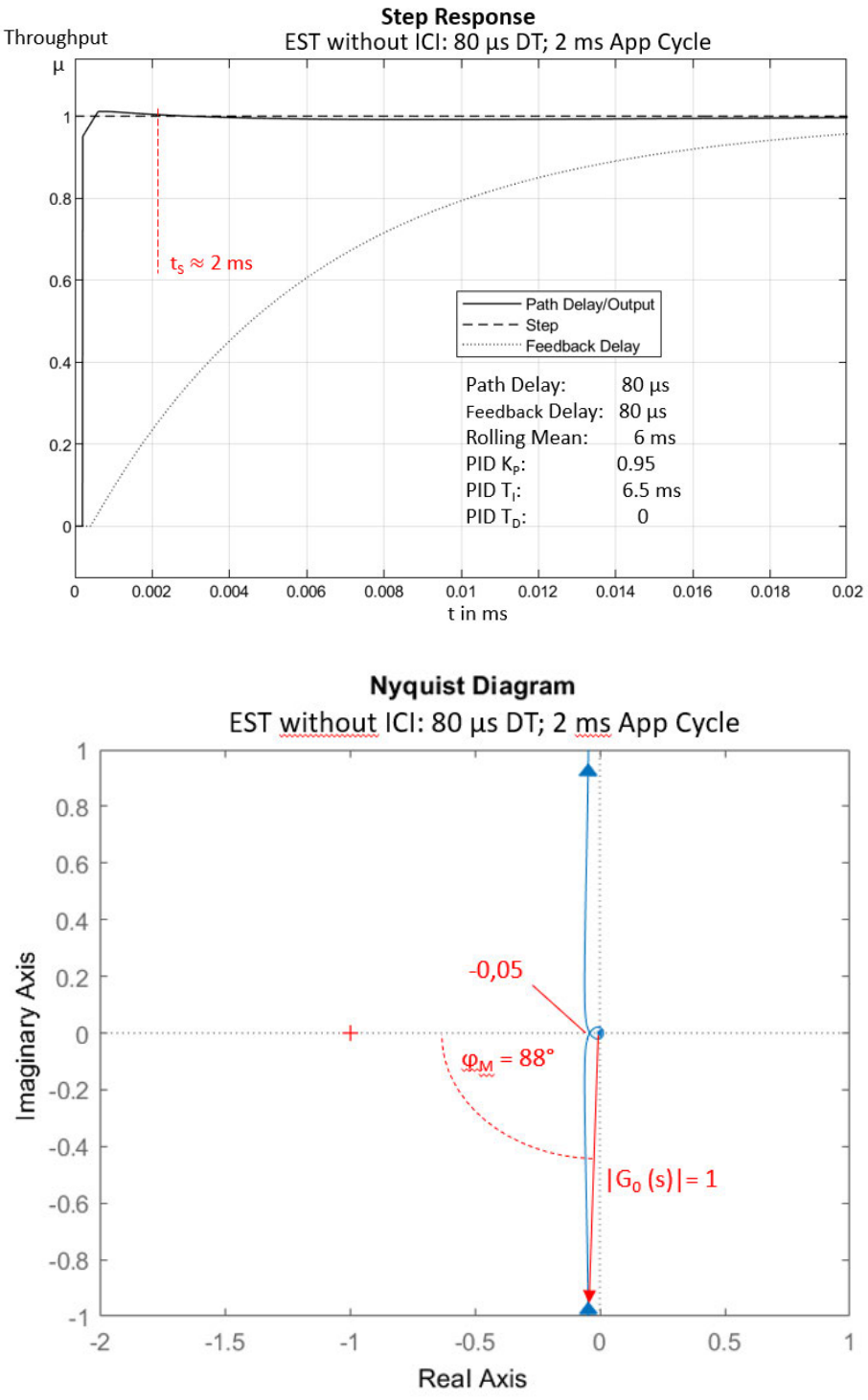


Figure 5.7: Step response and Nyquist diagram for EST

Figure 5.7 shows the step response and Nyquist diagram for EST without ICI representing the least possible dead time (DT or T_{DT}) solution of 80 μs for both path dead time and feedback dead time and thereby the network with the least dead time. According to Equation (5-23), with:

$$\tau = \frac{T_{DT}}{T_{DT} + T_{Mean}} = \frac{2 \times 80\,\mu s}{2 \times 80\,\mu s + 6000\,\mu s} = 0.03 \tag{5-31}$$

the control circuit is clearly lag dominant. The control circuit features a fast settling time of $T_S = 2\,ms$, a gain margin of $\frac{1}{0{,}05} = 20 \triangleq 26\,dB$, and a phase margin of about 88°, thereby representing a fast and robust control design.

Figure 5.8 shows the step response and Nyquist diagram for the SPQ with maximum ICI. It represents a control circuit with a medium dead time of 680 μs for both the path dead time and feedback dead time. According to Equation (5-23), with

$$\tau = \frac{T_{DT}}{T_{DT} + T_{Mean}} = \frac{2 \times 570\,\mu s}{2 \times 570\,\mu s + 6000\,\mu s} = 0.1\,, \tag{5-32}$$

the control circuit is still a lag dominant network. It features a quite fast settling time of $T_S = 10\,ms$, a gain margin of $\frac{1}{0{,}2} = 5 \triangleq 14\,dB$, and a phase margin of about 75°, representing still a rather fast and robust control design.

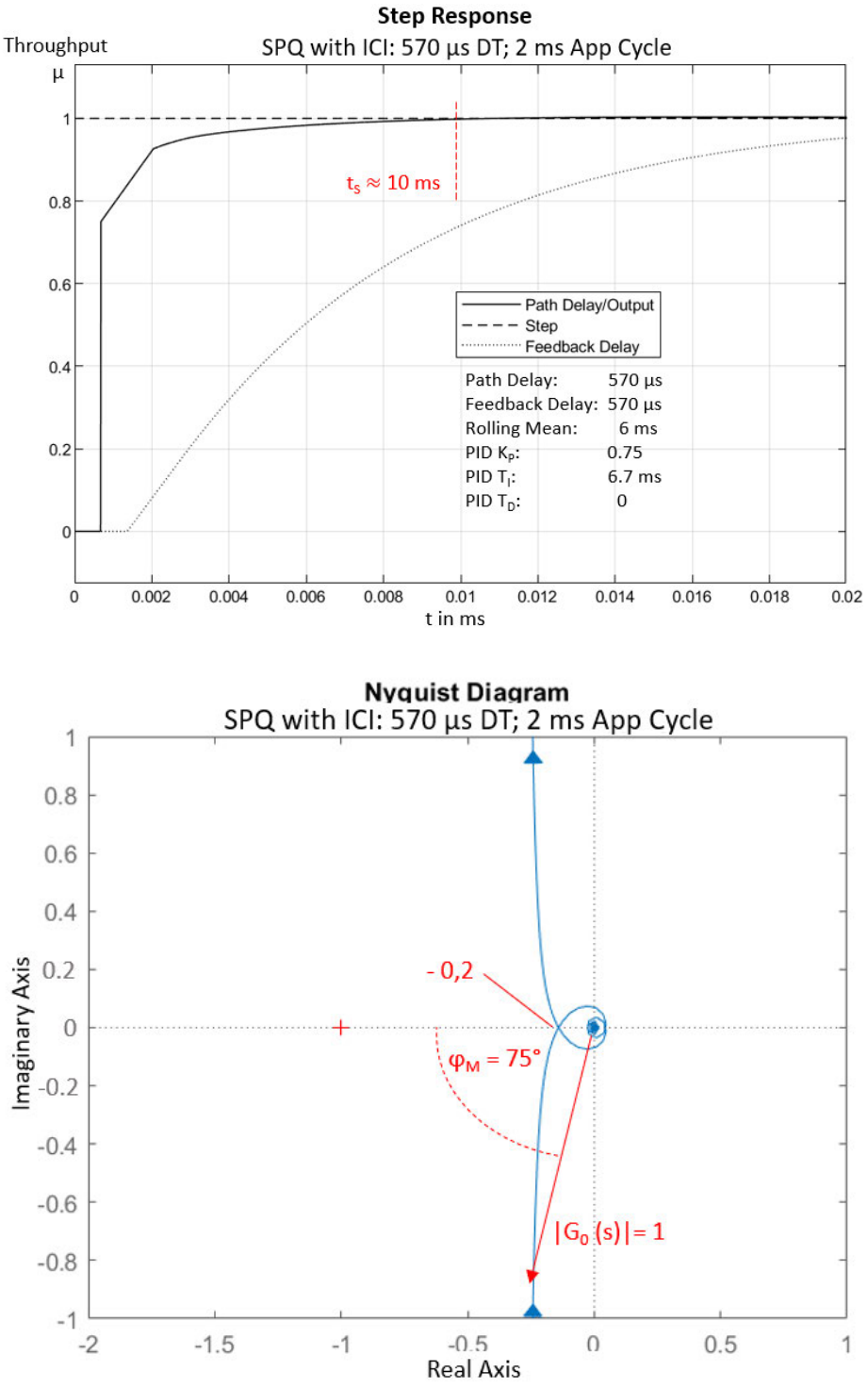


Figure 5.8: Step response and Nyquist diagram for SPQ with ICI

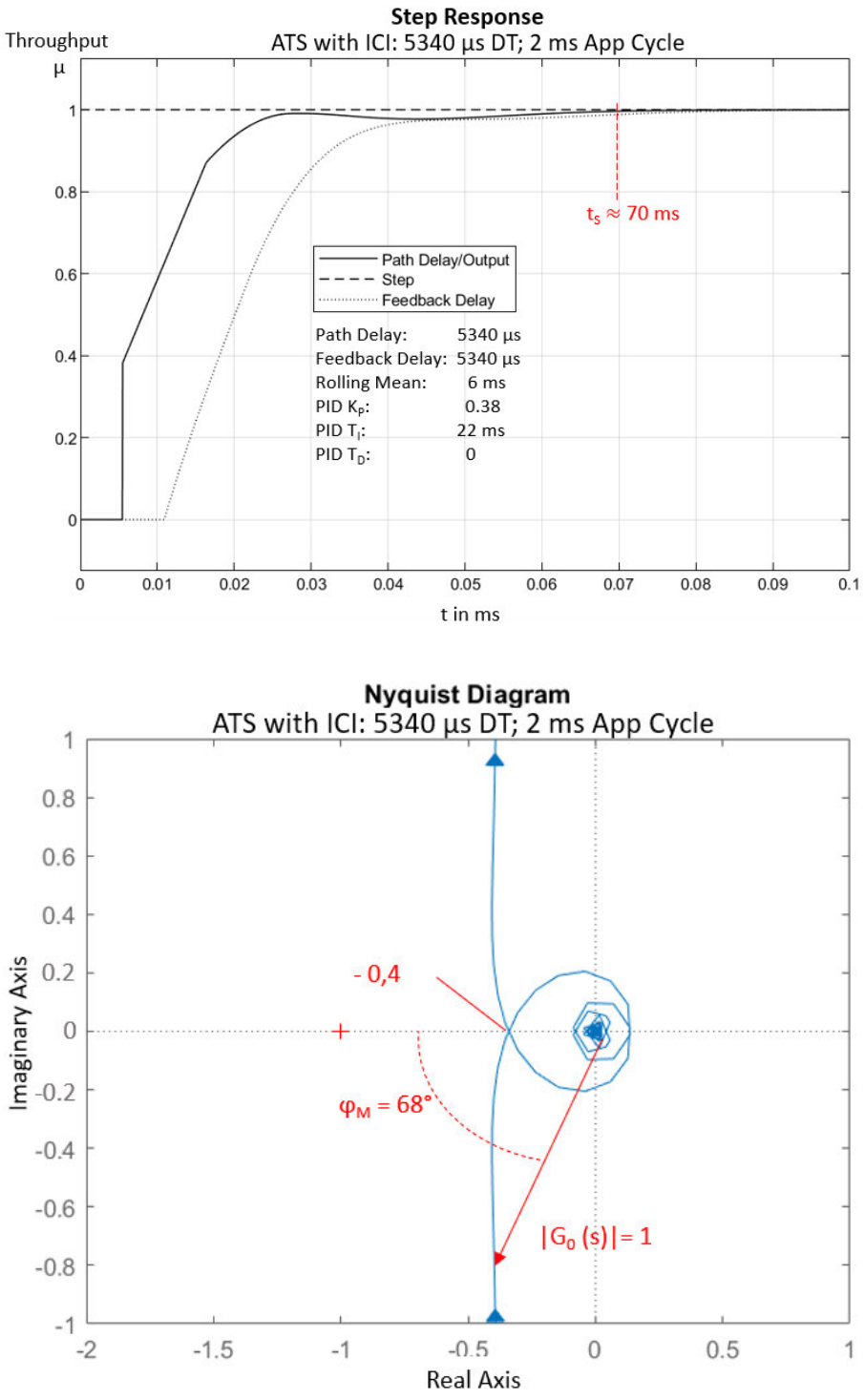


Figure 5.9: Step response and Nyquist diagram for ATS with maximum ICI

Figure 5.9 shows the step response and Nyquist diagram for ATS with maximum ICI representing the traffic shaper and traffic type with the worst dead time of 5340 µs for path dead time and feedback dead time. According to Equation (5-23), with

$$\tau = \frac{T_{DT}}{T_{DT} + T_{Mean}} = \frac{2 \times 5340\ \mu s}{2 \times 5340\ \mu s + 6000\ \mu s} = 0.64\,, \tag{5-33}$$

the control circuit is at the border of being dead time dominant. The settling time has worsened to 70 *ms*, the gain margin to $\frac{1}{0,4} = 2{,}5 \triangleq 8\,dB$, and the phase margin to 68°, representing a control design at the border of robustness.

Figure 5.7 to Figure 5.9 clearly show the influence of the path dead time and feedback dead time. With increasing dead time, the necessary control loops settling time t_S grows approximately proportional. At the same time, the intersection of the Nyquist diagrams with the negative real axis shifts with increasing dead times towards -1, which is the critical point for stability. This results in lower gain margins and lower phase margins and thereby less robust flow control circuits. Table 5.6 summarises the results.

Table 5.6: Simulation results for shaper/scheduler examples for a fast 2 ms application cycle dominated network.

Traffic shaper and traffic type	**worst case path dead time T_{DT} (μs)**	**settl. time t_S (ms)**	**gain margin g_M (dB)**	**phase margin φ_M (°)**	**ctrl robust-ness**
EST without ICI	80	2	26	88	high
SPQ with maximum ICI	570	10	14	75	medium
ATS with maximum ICI	5340	70	8	68	low

Because the dead time is either calculated or measured over an appropriate time span, the actual dead time can differ. The possible uncertainty in the dead time calculation or dead time measurement makes a tuned flow control circuit at least imprecise or even unstable. To illustrate the effect of a dead time deviation, Figure 5.10 (a) shows an example of the step response for the SPQ with 50 percent ICI.

A dead time of 470 µs was assumed for both path delay and feedback delay. The PID controller was optimised for *DT* = 470 μs dead time. In this case, the maximum deviation is represented by either no ICI or maximum ICI, leading to either *DT* = 370 μs dead time or *DT* = 570 μs dead time, respectively.

Figure 5.10 (a) shows that the effect of the error is only a slightly mistuned control circuit. In this case, it provokes a rather acceptable slower settling time of $t_S = 10\ ms$ for both 370 μs and 470 μs dead times compared to 8 *ms* for the 570 μs tuned control circuit. Figure 5.10 (b) shows the result when the same test case was applied to a tuned control circuit featuring the ATS. An average medium dead time of *DT* = 2,710 μs, a possible deviating minimum dead time of *DT* = 80 μs and a maximum dead time of *DT* = 5,340 μs were assumed. The considerably deviating dead times provoked substantial deviations in settling time of $t_S = 70\ ms$ for $T_{DT} = 80\ \mu s$, and $T_S = 95\ ms$ for $T_{DT} = 5{,}340\ \mu s$, compared to the tuned settling time of $t_S = 15\ ms$. In addition, the higher actual dead time produces a considerable overshot of 40 %.

Summarising the influence of ICI, it can be stated, as Figure 5.10 (a) shows, that ICI has very low influence on the control quality of SPQ. The small dead time deviation tolerance band of only +/- 20% has hardly any noticeable control performance consequences for the sample network. The tolerances for EST and CQF, summarised in Table 5.4, were also uncritical. However, ATS, as illustrated in Figure 5.10 (b), has a high uncertainty of nearly 90 per cent in this case. This is caused by the asynchronous gating times between the bridges, which results in poor settling times and a high overshot for the flow control circuit.

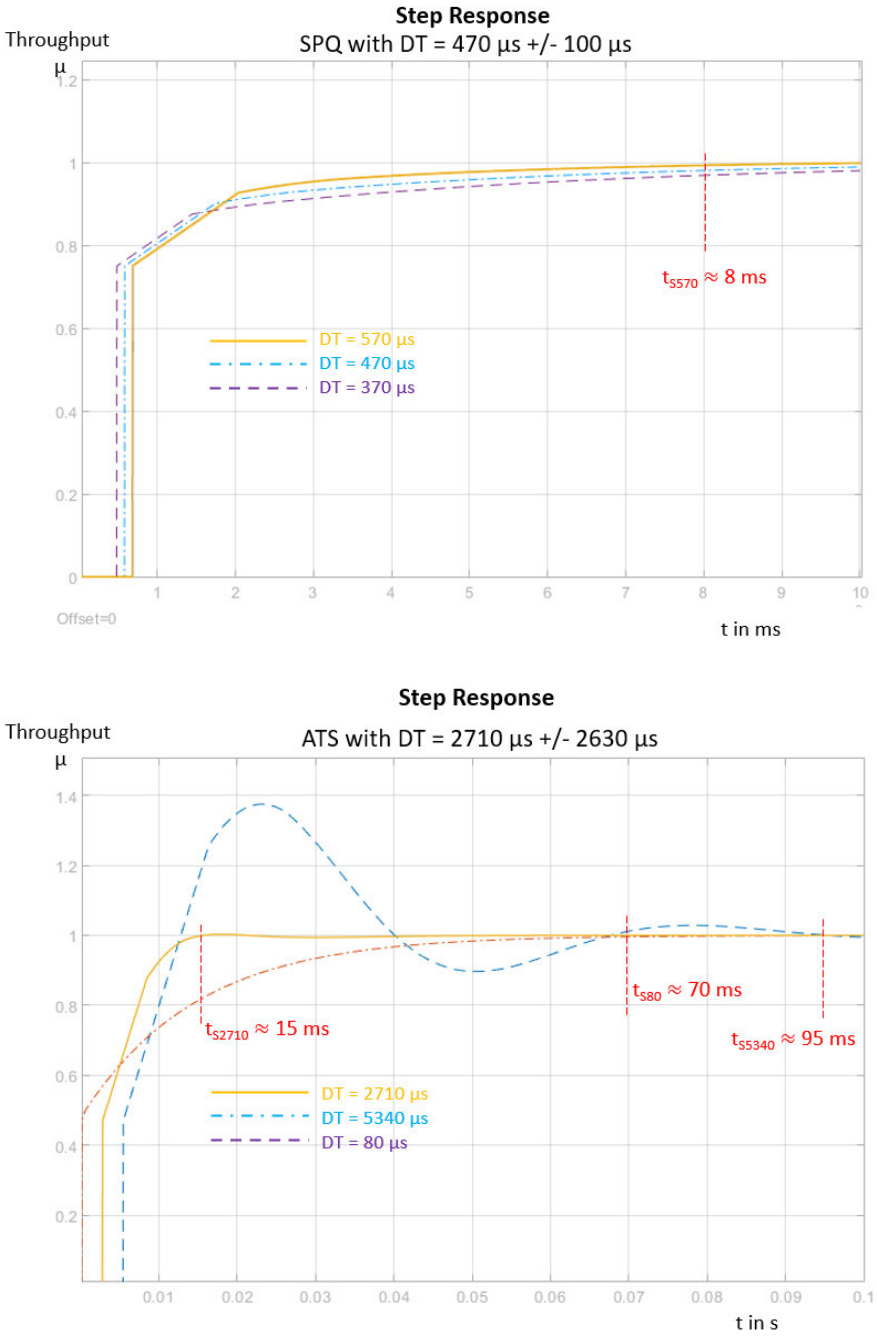


Figure 5.10: Dynamic performance deviation depending on dead time uncertainties

5.7 Chapter Summary

Different TSN traffic shapers introduce different dead times into the flow control circuit. It is shown that EST, SPQ with Preemption, and SPQ without Preemption are clearly the best selections from the flow control point of view because of their lower absolute dead times. CQF and ATS delay times increase stronger with

increasing amount of admissible data traffic and number of hops compared to EST and SPQ. Therefore, CQF and ATS could also be worthwhile selections within either smaller networks, lower loaded networks, or both. A further result is that the overall possible traffic load of the relevant traffic classes, that is, the ICI, on the path has an influence on the gating window size for EST, CQF and ATS and thereby on the resulting dead times. A higher possible ICI demands longer gating windows and means higher resulting dead times. For SPQ, ICI cannot be overtaken by Preemption and is therefore also a source of additional dead time for SPQ with Preemption.

The slowest application cycle within a traffic class assigns the minimum integration time of the rolling mean calculation of the throughput feedback and thereby the achievable dynamic performance of the flow control circuit. For very fast applications of a 2 *ms* application cycle time or faster, only EST and SPQ are recommended traffic shapers in combination with load control for extensive automation rings. The use of the slower traffic shapers CQF and ATS makes sense if the network domain contains slow applications of at least “slowest application time > two times longest path dead time” to avoid dead time dominant control circuits with a need for elaborate and CPU time-consuming controllers. If ATS is to be used in connection with flow control, the control circuits should be optimised by assuming maximum dead times to avoid overshot during operation with dead time variations in the direction of lower values.

A further outcome of the analysis is, that the ETS, in addition to its low absolute dead time and low dead time uncertainty, offers the unique possibility of separating data transport for fast applications from slower ones. This can be realised by implementing independent and dedicated gating windows for groups of applications. Therefore, tailored and decoupled flow control circuits can be implemented for different application groups, that is, fast dynamic load distribution control for fast applications, and slow load distribution control for slow applications.

A further problem addressed in this chapter is how to derive the actual plant properties to be in the position to determine the appropriate control parameters.

Three alternative methods were proposed. First, the identification during the design phase derives control parameters from the knowledge of all bridges and bridged end station delays and all path delays. Hints for possible path estimation possibilities are provided in combination with certain traffic shapers and schedulers. A second possibility is the use of test messages to empirically obtain the parameters at runtime. This method is especially at hand within timely unsynchronised networks. For synchronized networks, the third possibility, to derive the values from the time synchronization protocols is the most effective way.

As a next step, in the following chapter, the optimal distribution control in ring topologies for a variety of listeners depending on the location within the ring is proposed and validated. A further challenging task is to find a proper method for the collaboration of several controllers that apply LDC at the same ring, thereby influencing each other. This is addressed in Chapter 7 .

Chapter 6 A New Control Method for Load Distribution Optimisation in TSN MAN

6.1 Introduction

After the requirements for the task of load distribution control of TSN automation networks in their various forms have been clarified in Chapter 4 and Chapter 5 , the corresponding network requirements, core control mechanisms, and optimisations can now be defined.

For this goal, it would be illusory to believe in finding a completely new superior core control method in addition to the already established methods for approximately three decades, as described in the literature review. More obviously, it will effectively mean that the best core control mechanism suitable for MANs is to be selected and is then further optimised and extended for the application in MANs.

According to the literature review, a variety of load distribution or load balancing methods are available for networks such as ISP networks, campus networks, and the access networks of mobile networks. To analyse their applicability and their possible further expandability for automation networks, it is advantageous to assess them based on their different solutions for the control elements as introduced with Figure 2.3 in Subsection 2.3.2:

1. The controller type,
2. The load measurement or load calculation method,
3. The control goal,
4. The controller output intervention,
5. The controlled traffic type which influences the control variable.
6. The traffic cycle time selection (if any).

Each of these control elements should be decided upon and possibly optimised for the individual requirements of manufacturing automation networks.

6.2 Determination of Advantageous Network Preconditions

As shown in Chapter 4, there are various decisions to be made when designing an automation network in general, particularly if this should support dynamic load distribution.

First, it must be decided whether to use a central or distributed control concept because this decision already limits the available controllers. Although the central solution is best suited for reaching the optimum traffic distribution results, the distributed solution is selected here. This is because it is better suited for dynamic load changes, as outlined in Section 4.2, which is especially important in MANs that can experience network extensions by new machines or automation cells.

Regarding network topology, the ring topology is the prevalent topology in redundant MANs and is also the basic topology here.

As the distributed load distribution control concept is chosen, the load controllers shall be located on the ACs. As outlined in Section 4.4, the other possibility for the distributed control solution is that the load distribution control would be located on a bridge. However, this is rather an atypical task for a bridge and is therefore not considered here.

This chapter focusses on data distribution and data flow control for a single AC. Load distribution control in the case of multiple ACs is discussed in Chapter 7 .

The selection of the distributed LDC concept also has an influence on the core controller type selection, as not all types are suited well for the distributed control approach. This will be discussed in the following section.

6.3 Discussion and Selection of the Basic Controller Types

6.3.1 Introduction

As outlined in Section 1.2, different controller types have been applied in communication networks for both the overlaid load distribution control and subsequent flow control so far.

All the applied control types have particularities providing advantages and disadvantages in different types of networks. They can be applied as alternative methods to each other for certain applications within certain limits. To select an appropriate core controller type, the specifics of each controller type have to be analysed from the viewpoint of TSN MAN.

As Chapter 5 has already shown, with the strong variation of dead times, data traffic, and applications, one single best qualified controller core type cannot be expected for all variants of the TSN.

6.3.2 Discussion of the Relevant Controller Type Properties

A **linear controller** is the traditional option for regulating linear systems. This is the case for MAN, as they are analysed in this thesis, as outlined in Subsection 2.3.2. This is especially true for flow control to control the increase or decrease of the load of a path. However, depending on the relation between the delay that the data traffic experiences over the path and the necessary rolling mean integration time, the application of a predictor-based linear controller such as the Smith Predictor Controller or the Model Predictive Controller can be necessary. This is the case when the delay component is large compared with the integration time delay, as shown in Chapter 5 .

Stochastic network control is typically applied where incoming traffic arrival events are not known in advance, that is, they are stochastically distributed. However, a typical characteristic of automation network data traffic is that the data traffic volume and occurrence is usually well known, as it is planned to a certain extent. This is especially true for CD. Also, with CD, as outlined in Subsection 2.2.3, the process data are always sent at the beginning of an application communication cycle in the shape of a burst. Therefore, stochastic network load control is not the first selection for automation network control data. Its application would make more sense for data traffic created by applications sending data at irregular intervals or at intervals that are difficult to predict.

Predictor-based controller types, such as the Smith Predictor or the Model Predictive Controller are especially suited for coping with longer dead times within the data path or the feedback path (Normey-Rico & Camacho, 2007). As shown in Chapter 5 , this can be the case with higher-delay TSN traffic shapers such as ATS or CQF shapers in combination with fast application cycle times for CD traffic. However, in this thesis, the advantages of EST regarding its capability to decouple the data traffic of dedicated traffic classes are required. This ability will be used in a later stage of this thesis to decouple the data traffic from different automation controllers. As the EST traffic shaper has the characteristic of a comparably low latency time, as shown in Chapter 5 , and thereby a low dead time, there is no need to apply predictor-based controller types as a basis for optimised control within this thesis. This is also true for SPQ with a similar low latency time, which will also be applied in this respect.

The **fuzzy control** is especially advantageous when the system to be controlled is either rather complex, its behaviour is difficult to describe mathematically, or both (Pompili & Priscoli, 2008; Wang & Hung, 2012), as outlined in Subsection 2.3.3. This is particularly true when the system contains non-linearities. As the control plant, that is, the network paths and feedbacks of the automation communication network addressed in this thesis, are both linear and of limited complexity, the application of fuzzy control is not a need, but can be an alternative.

With **neural network control** a neural network (Hagan et al., 2002) is used either as a function approximator or as a neural controller, as introduced in Subsection 2.3.3. As outlined in Subsection 2.4.3, the neural network model within the neural controller predicts the plant response over a specified time horizon. The application of Neural Network Model Predictive Control is advantageous with non-linear and/or complex plant properties that are difficult to describe mathematically, which is not the case here. Automation networks behaviour is mathematically describable with limited effort owing to the linearisation of discrete data packets via rolling mean formation. But as is already the case with the fuzzy control method, for the networks investigated in this thesis, Neural

Network Model Predictive Control can be an alternative control method to the linear control method.

AI/ML controller selection is a further alternative for controlling complex systems or systems that are difficult to describe mathematically. As already shown with the assertion of the other traffic shapers and schedulers in this section, this is certainly not the case for the data flow controller selection. However, it could be a good selection for the optimisation task of distribution control to determine the most favourable overall load distribution. The learning phases of MLC can have a negative impact on the automation process though. Furthermore, it needs a high implementation and computation resources effort. For the single AC at a ring use case, it is from current point of view rather unlikely that the effort for an AI/ML control is worthwhile. Multiple AC in the ring with mutual interference are more likely to gain advantages with AI/ML. This case will be discussed in more detail in the next sub-sections and in Chapter 7 .

The **ant colony optimisation control (ACOC)** is especially suited for a distributed control approach, which is also considered for the optimised control concept in this work. The distributed approach is inherent in ACOC, as the agent packets need to be sent and received by the end stations making use of the control. According to Subsection 2.3.3 its advantages are scalability, distribution of the computational load, ruggedness to network errors, and its suitability for multiply meshed networks. The disadvantage, however, is the inclusion of the control layer in every routing decision and thus the exclusion of the TSN traffic shapers and schedulers. Other disadvantages are a possible temporal overuse of paths and the necessary high agent frequency to achieve high dynamics. Despite these disadvantages, the ACOC can, in principle, be used as a possible distribution controller method. However, it is not suitable as a solution for a subsequent flow controller which increases or decreases load on a single path.

6.3.3 Controller Type Selection Criteria for MAN

Considering the properties of TSN automation networks as derived in Section 4.6 and the controller type properties from the previous subsection, a few boundary conditions and consequences must be defined:

1. Load control is possible for BE and CD data traffic types. It is most advantageously applied to CD which is the focus of this study. CD is typically sent in a constant succession of data bursts. Measuring the throughput over a succession of burst cycles as a rolling mean value results in a linear throughput measurement. Therefore, LDC represents a linear control.
2. Lag dominant networks are assumed, that is, the dead time elements introduced by the path traffic shapers or schedulers are low compared to the time constants of the PT1 elements caused by the rolling mean calculation. The reason is that the SPQ and EST traffic shaper shall be used because of their low delay time and the EST traffic decoupling possibilities.
3. For distribution control, it is important to determine whether the control should be distributed and autonomous on one or more AC or whether it is to be located in a central instance for several automation controllers. This thesis focuses on the distributed approach for its dynamic advantages as outlined in Section 4.2. In the first place a single AC is considered. In Chapter 7 , the distribution control method is extended to several ACs in the automation ring.

The goal of this thesis is not to find the ideal type of basic controller core by an extensive comparison of all possible controller properties. This approach could easily fail because of the great variety of automation networks regarding automation application cycle time distribution, traffic types, and traffic scheduler and traffic shaper types. The goal is rather to select a well-suited core controller type that covers the most important and typical automation applications, and to extend the control principle to suit the LDC task in these environments. As will be visible further down, some core controller type could, in principle, also be exchanged with other controller types. For example, if a different traffic shaper concept would be selected for the data plane of the network this could introduce

a higher bridge delay and, thereby, a higher dead time. Then, it could be necessary to replace, for example, a selected classical PID controller basic control core by a Predictive Controller type such as the Smith Predictor Controller or the Model Predictive Controller.

6.3.4 Flow Controller Selection

First, the type of flow controller must be assigned.

An important question in the selection of both the flow controller and the distribution controller is whether abrupt changes in load on a path will cause the output to exhibit non-linear behaviour. This requires a closer inspection of the communication data as input and output variable.

The actual network consists of a pure dead time control element, as shown in Section 4.6. This dead time is caused by the LAN propagation delays and bridge latencies. The bridge latencies again depend on the applied traffic shaper or traffic scheduler of the data plane as discussed in Chapter 4 . So, a data bit, data byte, or data packet are just “shifted” through the network without any deformation. The peculiarity of data communication paths as controlled systems is that the input and output variables, that is, the data per time value, are non-linear when measured over a sufficiently small time span. This results from the fact that basically the data bits are transported serially over the path[1]. If the measuring interval would be reduced to a bit time, the input variable would actually jump nonlinearly between full and zero use. If the measuring interval is extended to a byte length, full bandwidth use is measured at sending times, but zero bandwidth use in the sending gaps. The same applies if the measuring interval is extended to frame length. If the sending characteristic is burst, as with automation applications CD as outlined in Section 4.7, all measuring intervals smaller than the

[1] Strictly speaking also modern serial transport mediums such as Ethernet cables have a certain extend of parallel transport as they contain several twisted pair strands.

burst repetition cycle times would deliver oscillating throughput measurement values, which also becomes clear from Figure 4.8.

From these considerations, it is obvious that a network transporting data measured per time over a sufficiently long time interval must be handled as a linear system, although a pure dead time element would have to be handled as a non-linear system. Therefore, for the flow control of CD, there is no need for dedicated controllers to cope with non-linearity or randomness of input data, such as stochastic control, fuzzy control, neural network control, or machine learning control.

Moreover, a dead time in the system to be controlled can be compensated by sufficient delay times in the control loop, as shown by Normey-Rico and Camacho (2007), and discussed and confirmed in Chapter 5 . In this case, such a network can most easily be controlled by means of a traditional linear controller in the shape of a PID controller.

Therefore, the second question is whether the network is lag dominant, that is, the delay times in the network path are low compared to the time constants in the feedback path, as analysed in Chapter 5 . This can be assumed with the use of the EST traffic shaper and SPQ being the focus of this thesis in combination with fast application cycles down to 1 ms, as outlined in Chapter 5 and the previous subsection. Therefore, no prediction-based controller types are necessary.

For these reasons, the classical PID controller is selected as the basic flow controller type.

6.3.5 Distribution Controller Selection

To be in the position to select the distribution controller type, the controller location must be firstly decided. Shall the control be located centrally on a CNC instance, or should it be located de-centrally on the automation controller? In the latter case it is furthermore important whether the control is independent of and unaffected by other distribution controllers of other ACs. In the previous subsection these questions were answered with a focus on the distributed control.

In the first step, this is located on a single AC without considering the possible influences of other distribution controllers.

A further prerequisite for the selection is that the actual load distribution optimisation method needs to be selected. As shown in Section 4.6, there are primarily two possible goals for the optimisation of the network load distribution in the foreground:

1. One conceivable optimisation goal is the **maximum-reduction** which means the minimisation of the maximum local load μ_{ij} until the maxima on both paths are equal or nearly equal within a predefined tolerance band.
2. An alternative goal would be the **optimum-distribution** to achieve a distribution that is as even as possible on all paths.

Note that a path selection for the minimum delay time from talker to listener would also be a worthwhile optimisation goal but is not a load distribution optimisation goal and is out of focus here.

To achieve the **maximum-reduction goal** for a single distribution control on one AC, the following tasks must be implemented:

1. The relevant data traffic bandwidth consumption must be measured over a suitable time span at each ring port in the network domain. For this purpose, the measured values of the subsequent flow control circuit can be reused.
2. The distribution control preprocessing of the distribution control assembly compares the maximum values of each path and feeds half of the difference to the distribution controller as a reference and from there, into the flow controller.
3. The flow control output is fed into the packet controller, which feeds both paths. For the direction with the current maximum, it will be fed negatively, and for the other path it will be fed positively.
4. The reference provision for the flow control can either be applied as a simple P-control that provides half of the maximum difference or can be further smoothed and possibly accelerated by applying additional integration and derivative control elements resulting in a PID-type distribution controller.

The **optimum-distribution goal** in combination with the distributed control concept requires a different approach:

1. Combined with the selected distributed control concept, it is difficult to achieve an *overall* optimum distribution. This is because a single controller requires special additional coordination mechanisms with all other distributed controllers. These additional coordination mechanisms are expected to be difficult to achieve, particularly with regard to the avoidance of load oscillations.
2. Therefore, for the distributed approach, a compromise *towards* an optimum distribution is selected that can work without high-effort ACs coordination. This is further elaborated on in Chapter 7 .
3. From application engineering, an AC has information about each of its own originating streams of a certain traffic class regarding the frame size, application cycle, registered listeners, and their distance from the AC in each direction of the ring. Thereby, it can calculate the bandwidth use contribution of each stream for each ring port of the ring network domain.
4. Each AC strives for optimum distribution of its own traffic in the ring. Thus, it can be assumed that the sum of the traffic of all ACs is also homogeneous to a certain extent.
5. Smaller end stations, from a traffic generation point of view, or end devices different from ACs, will not support their own distribution control for their transmitted traffic. Integrating these into dynamic traffic distribution control would typically only make sense by applying central traffic distribution control. Nevertheless, their traffic would be measured, recorded, and published at the ring ports in the same way as interfering exogenous and unknown traffic and can thus also be considered in the distribution control for optimum distribution. For integration into the distributed control, the end stations that are addressed by the AC will use the reception of CD as a trigger to start their sending process towards this AC. It is important for end stations without an own load distribution controller, that they do not change the initially selected paths for their CD towards ACs; otherwise, the measurement conditions would

constantly change for the AC. The initially selected paths chosen by the end stations can be selected using criteria such as the minimum accumulated delay calculated during the reservation process by the MSRP or RAP protocols. Another possibility is a pre-engineered path selected by the network design at the configuration time. If traffic is allowed to select a different path, the AC can provoke this by sending appropriate managing configuration to this talker. See Subsection 7.4.3 for further details.

6. A challenge is the use case of multiple ACs in or at the ring as it must be agreed upon among those via an additional algorithm in which the AC compensates for the data traffic of unknown ACs and small end devices. See Subsection 7.4.2 for a solution proposal as to this.
7. For a single AC, the distribution control task is to constantly calculate the optimal traffic distribution by applying a dedicated algorithm that computes the following steps:
    a. Calculate the traffic distribution for each ring link for each possible combination of path utilisation of the single streams under consideration of the measured non-controlled streams of unknown talkers from unknown ACs or from smaller end devices.
    b. Calculate the least squares deviation of the mean traffic bandwidth use for each possible stream-path use combination.
    c. Select the optimal stream distribution or the first stream distribution that is within the acceptance tolerance band.

Comparing the two distribution control concepts for maximum reduction and for optimum distribution, it is obvious that the optimum-distribution concept requires much more effort for the single AC than the PID controller for the maximum-reduction optimisation goal. Furthermore, the maximum-reduction goal is of higher importance because it can provide an immediate reaction to a possible loss of data in the case of local bandwidth maxima when it is near the maximum bandwidth. Therefore, the maximum-reduction goal within the influence area of a single controller is regarded as a better optimisation goal. This is especially true in combination with the EST traffic shaper, which allows the

traffic of a single AC to be decoupled by assigning dedicated EST windows to each AC. Thus, each AC sends and controls its traffic at dedicated point in times, avoiding interference with other ACs.

Consequently, the most advantageous distribution controller type basis for the controller optimisation to build upon is the linear PID controller in combination with the maximum-reduction optimisation goal.

As will be shown in Chapter 7 and in more detail in Subsection 7.4.2, it will also be possible to apply the maximum-reduction method to multiple mutually dependent ACs in order to iteratively get closer to the optimum-distribution goal.

6.3.6 Discussions and Evaluations

To date, various controller types have been applied in general communication networks for both overlaid load distribution control and subsequent flow control. These are mainly linear control, stochastic control, predictive control, ant colony optimisation control, neural network control, machine learning control, fuzzy control, and dedicated control algorithms. All methods rely on the measurement of the data throughput per time, which means counting data arrival events over a suitable time span. The result is always an average or mean calculation of the packets or data frames per time. It is a measurement of a continuous value over the complete input and output range, and thus the network can be handled as a continuous and linear system. Therefore, there is no need to apply elaborate control methods for nonlinear systems such as fuzzy, neural or machine learning control. Since the applications CD is also sent at regular and known intervals, no stochastic control is required. Furthermore, the small dead times involved in connection with the EST traffic scheduler, which is the focus of this thesis because of its traffic decoupling capabilities, eliminate the utilisation of predictive controllers. This is also true in connection with the SPQ, which serves as an alternative solution in this study. These preconditions lead to the following decision overview, as listed in Table 6.1.

Table 6.1: Possible Distribution Load Control methods.

Control method	Application area	Possibility for flow controller for automation CD	Possibility for distribution controller for automation CD for single AC	Possibility for distribution controller for automation CD for multiple AC
Linear (PID)	Lag dominant linear systems	yes	yes	Yes, for distributed controller. Not for central solution
Linear Smith Predictor or Linear Model Predictive	Dead time dominant linear systems	Yes, in connection with slow traffic shapers/schedulers and fast application cycles	Yes, in principle, but not recommendable as linear PID is better suited for path difference calculation with small dead times only	Yes, for distributed controller. Not for central solution
Stochastic	Unpredictable traffic patterns, stochastically distributed	Yes, in principle, but not recommendable as linear is better suited for being faster	Yes, in principle, but not recommendable as linear is better suited for being faster	Yes, in principle, but not recommendable as linear is better suited for being faster
Neural Network	Non-linear and/or complex systems	Yes, in principle, but not recommendable as linear is better suited for network path is linear and non-complex	Yes, in principle, but not recommendable as the network model needs constant adaptations depending on the single links load changes.	Yes, in principle, but not recommendable as the network model needs constant changes depending on the single links load changes.
Machine Learning	Non-linear and/or complex systems	Yes, in principle, but not recommendable as linear is better suited	Yes, in principle, but not recommendable as the trained network model needs constant adaptations depending on the single links load changes.	Yes, in principle, but not recommendable as the trained network model needs constant adaptations depending on the single links load changes.
Dedicated Algorithm	Unpredictable traffic patterns, stochastically distributed	Yes, in principle, but not recommendable as linear is less effort	Yes, in principle, but not recommendable as linear is less effort.	Yes. One of the few feasible solutions to find an optimised distribution of traffic
Fuzzy	Unpredictable traffic patterns, stochastically distributed	Yes, in principle, but not recommendable as linear is less effort	Yes, in principle, but not recommendable as linear is less effort	Yes. One of the few feasible solutions to find an optimised distribution of traffic
Ant colony optimsation	Multiply meshed networks such as complex networks, mobile	No, as it is only deciding which path to use. No flow control	Yes, in principle, but not recommendable as linear is less effort.	Yes. But with inherent tendency to overload the found best path.

Control method	Application area	Possibility for flow controller for automation CD	Possibility for distribution controller for automation CD for single AC	Possibility for distribution controller for automation CD for multiple AC
	networks, data centers	possible.		

These evaluations recommend linear traditional PID control as the basic core controller for both distribution control and subsequent flow control for automation CD. The distribution and flow controllers are set up in a classical controller-cascade, with the distribution controller feeding the flow controller. This basis serves as the starting point for optimisation to achieve the goals of this thesis.

6.4 Analysis of Drawbacks of Current Basic Distribution Control Possibilities

The classical approach with load balancing examples from the literature review for measuring the load on a path is that all traffic is handled equally. Its load influence is measured together, regardless of the type of traffic, its send intervals, and its importance. Applying this method also to MANs would have the following disadvantages:

1. All traffic is assumed to have the same deterministic requirements. Unimportant traffic, which would have no problems with some delays, is given the same importance as CD of highly deterministic requirements.
2. There is no differentiation between sporadic and cyclic traffic. This is an obstacle to finding a possible load measurement integration interval without oscillations, as outlined in Sections 4.6 and 4.7.
3. There is no differentiation between frame repetition intervals in the case of cyclic traffic. This hampers the selection of the optimum load measurement integration interval. Automation CD is usually sent in the form of a burst at the start of an automation application cycle. It creates an inhomogeneous traffic pattern when viewed over a period smaller than the application cycle. If a

common throughput measurement is applied that does not differentiate between different application cycles, the rolling mean measurement interval must be adapted to a value at least equal to the slowest application cycle within the relevant network domain. Otherwise, the measurement provided oscillating load distribution results. Similarly, sporadic non-CD traffic would spoil the selection of a measurement integration interval if controlled together with CD. A further consequence of this fact is, that in dynamic network setups, where automation controllers, devices, or I/O extensions can be added or removed during operation, a constant adaptation of all automation controllers load control parameters to the slowest application cycle is necessary. This is particularly bad in an environment where new automation processes might be added or removed on the fly during runtime, which is a common requirement for most automation networks. Removing ACs with slow applications or disabling slow applications, has in the first place, not the important consequences of adding slow applications. The quality of load distribution control does not deteriorate. However, the control could be more dynamic towards an optimal control if the control parameters are adjusted after removal. The consequences of slight control dynamics deteriorations owing to the influence of the presence of slow applications might not be as far-reaching when the timely difference between fast and slow applications is small. However, with higher ranges of application cycle times, the disadvantages for the fast applications increase. This becomes clear immediately when considering fast motion control circuits with cycle times as low as a few microseconds. Their LDC dynamics can be tremendously slowed because of the presence of one temperature control with an application communication cycle of, for example, a few hundred milliseconds.

6.5 Proposal of a Control Method for Optimising Load Distribution for TSN MANs

As discussed in Section 6.3, for this research within MAN, a classical linear PID controller is selected as the core mechanism for both the basic distribution controller and subsequent flow controller. Both controllers, together with the feedback calculation and packet controller, represent the distribution-control assembly.

The current solution possibilities for load distribution control, as outlined in the literature review, are not prepared for application in MAN. As outlined in the previous section, they do not take the circumstance into account that application control data is sent in different application cycles. The optimised control method should be dedicated to the important traffic in the MAN, which are the CD streams. Therefore, the first step is to exclude non-CD traffic which is of secondary importance and can be controlled separately in a separate gating window. To overcome the drawbacks of a straightforward overall load distribution control as outlined in Section 6.4, the distribution control assembly is further supplemented and extended specifically for TSN MAN CD by the following features:

1. The first and most important is that, instead of a common load distribution control, a series of distribution controllers process dedicated application classes. The application classes are categorised based on their application cycle times. Their throughputs are measured individually per application cycle or application cycle group and are individually fed back to their dedicated distribution controllers.

$$\mu_{ij,CD,App\,\alpha} = \sum_{q \in \boldsymbol{Ta}} \mu_{ij,CD}\,(s^{q\alpha})\ ;\ \boldsymbol{Ta} \subseteq \mathbb{N}\,;\ \alpha \in \mathbb{N} \qquad (6\text{-}1)$$

The maximum-reduction optimsation goal is then changing from the form in Equation (4-3) to:

$$\min \max_{i,j \in \boldsymbol{V}} \mu_{ij,App\alpha}\ \text{; Subject to: } \forall\ e \in \boldsymbol{E}(G), \alpha \in \boldsymbol{A} \qquad (6\text{-}2)$$

In Table 4.4, the pseudo code parts:

```
For application cycle α <= SUMAPPSIND
{
  For direction j <= 2
    {
    For node i <= maximum number of nodes
      {
        If (m_thp_array [α] [j] [i] > max)
        {
         max = m_thp_array [α ] [j] [i];
  }
      }
      Store max at index number of nodes NNODES:
      m_thp_array [α] [j] [NNODES]= max;
      max = 0;
    },
```

and:

```
For application cycle α <= SUMAPPSIND
{
  For direction j <= 2
    {
    For node i <= maximum number of nodes
      {
        If (m_thp_array [α] [j] [i] > max)
        {
         max = m_thp_array [α] [j] [i];
  }
      }
      Store max at index number of nodes NNODES:
      m_thp_array [α] [j] [NNODES]= max;
      max = 0;
    }
```

determine the application-cycle-specific load maximum for each ring direction and store it at the array index NNODES for later application-cycle-specific distribution control. The optimum-distribution optimsation goal changes in the same way from the form in Equations (4-5) and (4-6) to:

$$\min \sum_{\substack{i,j=1 \\ i,j \in \boldsymbol{V}}}^{n} (\mu_{ij,App\alpha} - \mu_{M,App\alpha})^2 \text{ ; Subject to: } \forall\, e \in \boldsymbol{E}(G), \alpha \in \boldsymbol{A} \tag{6-3}$$

$$\mu_{M,App\alpha} = \frac{\sum_{i,j=1}^{n} \mu_{ij,App\alpha}}{2n} \text{ ; } n \in \mathbb{N} \tag{6-4}$$

2. The distribution control assembly sets up a delay list sorted by ring direction and ring node number. This is necessary to feed the actual distribution and flow controller with the optimum PID control parameter for optimum control

performance, depending on the location of the current load maximum. The delay values can be obtained by applying the different plant characteristic identification mechanisms proposed in Section 5.4.

3. Slower application cycle traffic can be intentionally excluded from distribution control because of its low overall balancing contribution. The consumption of bandwidth, or in other words, the caused throughput on a network link, depends on the packet length and application cycle. The shorter the application cycle T_{App} and the longer the packet length n_{BytesApp} , the higher is the applications bandwidth consumption μ_{App}. Equation (6-5) reflects this for the underlying maximum bandwidth of 1 Gbit/s:

$$\mu_{App} = n_{\mathrm{BytesApp}}\ Byte\left(8\frac{Bit}{Byte}\right)10^{-9}s\ \frac{1}{T_{App}} \tag{6-5}$$

   Thus, controlling the load distribution of faster applications contributes more to a balanced overall distribution than controlling slow application cycle classes. The system designer can weigh the requirements for the grade of load balance quality against the calculation and configuration effort.
4. Traffic from single talkers with a small amount of traffic without an own LDC can also be excluded from the distribution control for the same reasons as for the slow applications mentioned above. Alternatively, the assigned AC can choose the direction for this talker thus integrating it into the load distribution control.
5. The summation point of the path loads differentiation in the distribution control assembly on the AC is provided with a threshold to avoid small oscillations around the current working point.

The advantage of using dedicated controllers for groups of application cycle classes is the ability to mask out the impact of the slower applications in the network domain on the achievable control performance for faster application cycles. Furthermore, a possible later addition of network participants communicating at slower application cycles has no influence on LDC as their low influence is deliberately disregarded.

The typical automation network field-level ring consists of only one leading AC that controls several field devices, such as IO peripherals, drives, or distributed automation

peripherals. Therefore, it can be advantageous to implement a “ring-central” distribution control only on this AC neglecting smaller talkers that also put load on the network ring. Basically, it is not worthwhile to implement distributed load distribution units on talkers, which contribute only a small amount of traffic. But if these are under control of an AC, this AC can after its own load distribution calculations provoke them to use a certain ring direction to achieve a load shift on certain links.

In contrast to field-level rings, controller-level rings typically have multiple controllers. The problem with multiple ACs is that distributed dynamic LDC creates mutual influence over the common communication ring. To achieve a dynamic load distribution in this case, either a central control approach with a dedicated load distribution algorithm or a distributed solution with measures for mutual coordination or mutual decoupling must be considered. Refer to Chapter 7 for the proposed solutions in the case of multiple AC with distributed distribution controllers residing on each AC.

The reason for the proposed threshold at the differentiation point before the distribution controller is to avoid constant path changes with low traffic changes, which would result in unwanted path-selection oscillations.

Although the design of an appropriate packet controller for dedicated distribution controllers shall not be the focus of this thesis, some important influences and possible limitations should be highlighted here. An issue with separated application cycle load distribution controllers is that interferences can occur in certain application cycle classes without any original traffic occurrences within this application cycle class in the controlled automation network domain. A packet controller must then decide whether to adapt the traffic of another application cycle class or ignore such traffic situation occurrences with the consequence of tolerating a certain amount of load differences within an application cycle class. A further limitation to be considered with the EST and CQF traffic shapers is that the maximum possible bandwidth assignment possibility for CD is limited by the assigned EST and CQF network cycle length. The consequences of this fact are elaborated further in Subsection 7.4.3.

6.6 Discussion and Selection of the Optimised Feedback Method

As the literature review shows, different solution approaches exist to feed the distribution control assembly with information regarding the load situation on the links in the network. Farahmand et al. (2005) investigated congestion control on optical burst links and proposed a method in which the feedback signal specifically notifies the source of how much it should reduce its rate to match the targeted congestion level of the network. This is a different concept in which the actual controller is located on each node whereas the AC would only serve as a packet controller. However, this solution neglects to consider the dead times from the controller to the nodes, which are necessary for an optimally tuned flow controller.

The MATE adaptive traffic engineering solution (Elwalid et al., 2002) uses the end-to-end delay on the different paths as feedback for the control loop. The intention here is to use end-to-end delay as an equivalent of congestion caused by load. This works if it can be assumed that the paths from the source to the end have nearly equal lengths. However, this typically is not the case with ring networks. Here, in the worst case, when a node in question is right next to the AC, one direction is the complete ring, and the distance in the other direction of the ring is only one link. This already leads to greatly deviating path delays purely on the location of the node, without delays due to the traffic load also being taken into account.

Therefore, for automation network rings, the goal must be to measure the actual load on each ring port of the nodes in the ring. A further goal must be to consider the individual dead time from the distribution control assembly in the AC to the node with the current load maximum, to adapt the controller properties to the current control situation. Applying closed-loop control methods that work with maximum bandwidth use gained by path measurement and delay calculation or measurement avoids load oscillations. These would occur with mere congestion control using only acknowledgement losses, where path delays are not considered. Oscillations are an important disadvantage. They particularly come to effect in high bandwidth delay product networks such as ISP networks with high bandwidth and data transported over longer distances, as described for XCP (Kandula et al., 2005).

The optimised control method, as proposed in Section 6.5, has additional consequences for the formation of the feedback values for the control loops. As explained and applied in Chapter 5 , for a single flow controller, the feedback value must be measured over a suitable time span to avoid oscillations in the value. Therefore, in Chapter 5 , this time span is selected as a multiple of the slowest application cycle time in the network domain. With the new optimised control method, dedicated control loops, for each of the different application classes cycle times or for groups of classes, shall be applied. Therefore, also the mean values must be measured, calculated, and fed back individually for the application classes or groups of application classes. In the AC hosting the load distribution controllers, a maximum load value selection must take place to select the individual maximum throughput of the path per application cycle class.

Figure 6.1 depicts the optimised feedback creation process for one network path.

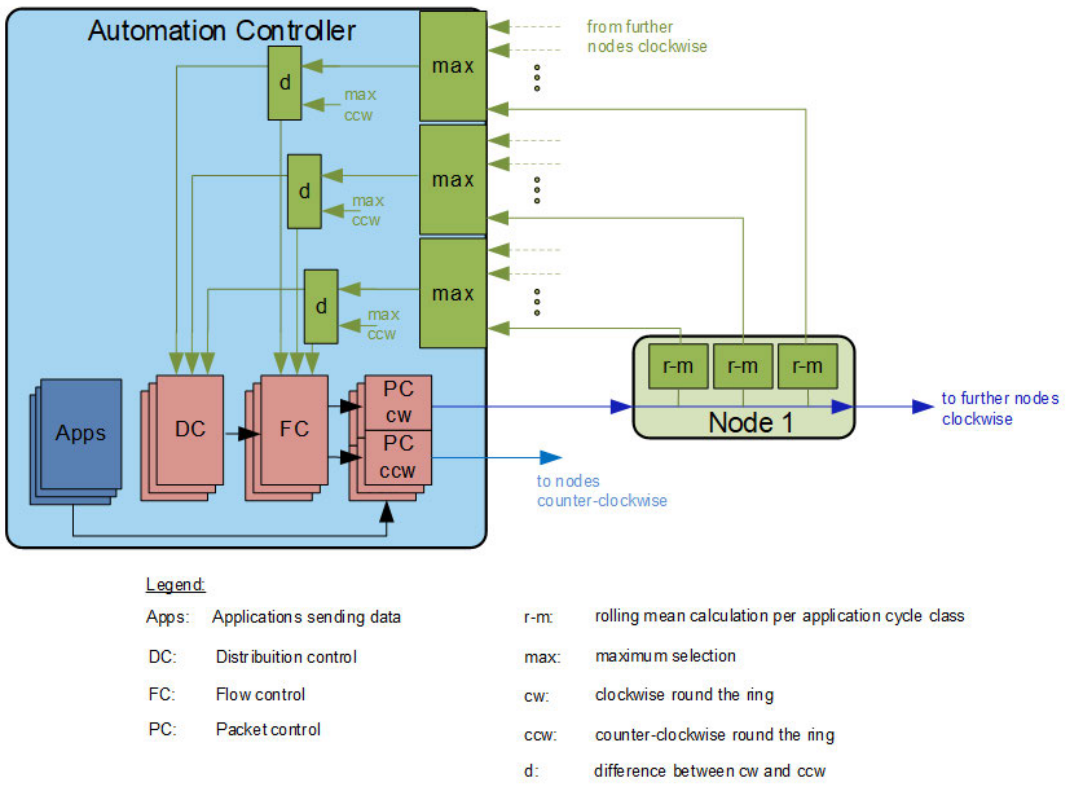


Figure 6.1: Optimised feedback creation process for one path

The actual transport of the individual rolling mean values can be either in individual frames from the nodes to the automation controller or by a collecting round-robin

frame constantly circling round the ring to collect the individual rolling mean values. The collection frame would have to be circled once per fastest application cycle rolling mean value measuring time span. Each node inserts its throughput measurement in an array organised by application cycle. The array is then written to the round-robin frame with a special offset for each node. This has the advantage of creating less throughput consumption, but the disadvantage is that new ring participants will cause a recalculation and reconfiguration of the offset for each node. For the network simulation in Section 6.7, the single frame solution was selected owing to its easier network extension capabilities.

Further information to be part of the feedback frame is the measured path delay to that node and port. This is needed for the optimum tuning of the distribution controller and the flow controller, as outlined above and in Section 6.5.

6.7 Performance Validations of the New Optimised Control Method

6.7.1 Introduction

To obtain the primary data for the validation of the new optimised control method introduced in the previous sections, an automation network in ring topology is simulated using the discrete-event network simulator ns-3. The simulator ns-3 is an open-source network simulator implemented in C++ and Phyton and is freely available at the Git repository https://github.com/nsnam under the GNU license conditions. Ns-3 has been widely used by the communication network research community. Ns-3 allows a simulation time resolution in nanoseconds for creating, time-stamping, and analysing communication events.

For the performance evaluations of the optimised control method, a single automation controller (AC1) network setup is simulated. The automation controller output is implemented by two applications sending frames clockwise (cw) and counterclockwise (ccw) into an automation ring network. This consists of nine virtual bridged nodes, n1 to n9, simulating either real bridges or bridged automation devices

containing listeners and/or talkers (L/T). The automation controller itself does not contain a bridge instance to save the application of redundancy protocols in parallel. Otherwise, broadcast data such as ARP requests and responses for address resolution would circle endlessly in the ring, creating a loop and thereby an avalanche of circling data. Figure 6.2 depicts the setup of the simulated automation ring network.

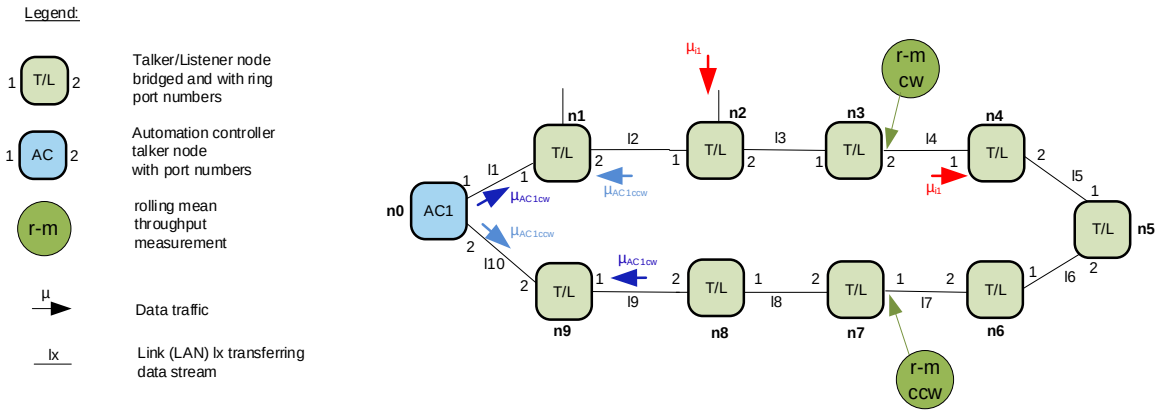


Figure 6.2: Automation ring setup for network simulations for performance evaluations

The automation controller hosts several applications instantiating several virtual talkers, which cause the overall load of streams (blue arrows). Multiple listeners residing in the ring nodes are assumed for all streams. Therefore, all streams from AC1 are sent around the complete ring in their individual directions. Without any exogenous interfering traffic from the inter-ring links, the automation controller divides the load equally in the two ring directions. To test the distribution control, an additional interference load μ_{l1} is injected at certain times at node n2, bound for a virtual listener at node n4. This causes an asymmetrical load distribution in the ring which is compensated by distribution control. The measured load difference is fed to the distribution controller, which then provides the reference for the flow controller. Depending on the algebraic sign of the distribution controller output, the flow controller either increases or decreases traffic in the clockwise direction in the ring. The counterclockwise throughput is then decreased or increased contrarily by the same amount of traffic. Both the distribution and flow controller are implemented as PID controllers as discussed in Section 6.3.

For each application cycle class, there exists a dedicated output queue in which the local applications of this application cycle store their frames either synchronously or asynchronously at sending time. This is simulated by assuming a sufficient number of applications to achieve the necessary bandwidth consumption for the simulation. The packet controller outputs send a portion of the frames in each direction according to the relation preset by the distribution and flow controllers. The individual number of frames to be sent in either direction per network cycle is calculated using the individual stream frame lengths of all reserved streams and the number of reserved streams of that application cycle class. Streams of nearly equal length are assumed to reduce the complexity of the simulation. However, in real applications, different stream lengths are to be expected. A sophisticated mechanism for the packet controller, to map the flow controller output to the packet controller output, considering different stream lengths and single or bundled streams, could be subject to further research but is not the focus of this thesis. The pseudo code for a packet controller for fixed stream lengths is outlined in Table 6.2. Its detailed structure is provided in Appendix 2.

Table 6.2: Pseudo code of algorithm for packet control.

Algorithm: LDCApp::Control ()

This algorithm of the LDC application in an AC connects the minmax differential throughput from Table 4.4 to the distribution controller and this to the flow controller. It converts the flow controller output into a number of packets to transmit. This method must be called cyclically with distribution controller and flow controller.

```
Create variables:
m_diffthroughput;     //difference of throughput
m_distctrlin;         //input for distribution control
m_distctrlout;        //output from distribution control, input for flow control
m_flowctrlin;         //flow control input
m_flowctrlout;        //flow control output, to be translated into m_nPackets for send unit
m_packetSize;         //packet size of packets to transmit
m_nPackets;           //number of packets to transmit
m_deltaPackets;       //difference of packets to transmit

Fetch the maximum detection output from throughput array (see Table 4.4):
m_diffthroughput =  m_thp_array [α] [CW] [NNODES + 1];

Connect this to the input of the distribution controller:
m_distctrlin = m_diffthroughput;

Create the flow controller input from the distribution controller output and the throughput
difference:
m_flowctrlin = m_distctrlout - m_diffthroughput;

Calculate packets and change algebraic sign as a positive difference means a reduction for
this direction:
m_deltaPackets = m_flowctrlout/(0.0001 * m_packetSize * 8);

Adapt the number of packets to be sent by the delta:
m_nPackets = m_nPackets + m_deltapackets;
```

To show the control problem and the effects of the optimised control method, a series of use-cases is simulated. Simulations start with a single control of a single fast application class. Further application classes of slower application cycles are added, and different feedback rolling mean integration times are applied. Finally, control circuits for dedicated application classes are added and the improvements become visible. Table 6.3 provides an overview of these use cases.

Table 6.3: Load distribution simulation use cases overview.

Use case	Application cycles (ms)	Interference cycles (ms)	Flow control? Type?	Distribution control? Type?	Rolling mean integr. time (ms)	Purpose/Comments
UC1	1	1	No control	No control	1	Basic use case without load control
UC2	1	1	Yes, PID	Yes, Basic, only Proportional	1	Flow controller plus limited distribution controller. The distribution controller works only as a proportional controller.
UC3	1	1	Yes, PID	Yes, PID	1	Flow controller plus full distribution PID-Controller
UC4	1, 2, 4, 8	1	No	No	1, 2, 4, 8	This use case shows the influence of the occurrences of application cycle classes on the control.
UC5.1	1, 2, 4, 8	1	Yes, PID, optimised for 1 ms app cycle.	Yes, PID, optimised for 1 ms app cycle.	8	Occurrence of slower application cycle classes without controller parameter adaptations
UC5.2	1, 2, 4, 8	1	Yes, PID, optimised for 8 ms app cycle.	Yes, PID, optimised for 8 ms app cycle.	8	Occurrence of slower application cycle classes with controller parameter adaptations
UC6.1	1, 2, 4, 8	1, 2, 4, 8	No control	No control	8	Multiple application cycle classes and multiple controller basic use case visualising load interferences without load control
UC6.2	1, 2, 4, 8	1, 2, 4, 8	Yes, PID, optimised for 8 ms app cycle.	Yes, PID, optimised for 8 ms app cycle.	8	Multiple application cycle classes and multiple controller basic use case visualising load interferences with common load control
UC6.2	1, 2, 4, 8	1, 2, 4, 8	Yes, PID, optimised for 8 ms app cycle.	Yes, PID, optimised for 8 ms app cycle.	8	Multiple application cycle classes and multiple controller basic use case visualising load interferences with common load control
UC6.3	1, 2, 4, 8	1, 2, 4, 8	Yes, PID, optimised	Yes, PID, optimised for	32	Like 6.2 but additional slow application cycle

			for 8 ms app cycle.	8 ms app cycle.		interference without controller parameter adaptations. Leads to control quality deterioration.
UC6.4	1, 2, 4, 8	1, 2, 4, 8, 32	Yes, PID, optimised for 32 ms app cycle.	Yes, PID, optimised for 32 ms app cycle.	32	Like 6.3 but with controller parameters optimised for slowest 32 ms application cycle class.
UC6.5	1, 2, 4, 8	1, 2, 4, 8, 32	Yes, PID, optimised for 32 ms app cycle.	Yes, PID, optimised for 32 ms app cycle.	32	Load change settling time for common load control in dependency of application cycle and of slowest application cycle
UC7	1, 2, 4, 8	1, 2, 4, 8, 32	Yes, multiple PIDs, optimised individually for app cycles.	Yes, multiple PIDs, optimised individually for app cycles.	1, 2, 4, 8, 32	Load control with application cycle dedicated load controllers.

The ns-3 simulation framework makes wide use of the C++ object aggregation. The class "object" forms the basis where applications, nodes, net devices, and communication channels, that is, links and sockets, are based on and created via object inheritance. For the simulation topology, nodes were created and net devices, standard applications, and dedicated applications were added to these nodes. Links were added to the net devices, and thereby, nodes were connected to a network. Packet transmission and processing are typically performed using dedicated application objects. To provide the necessary functionality for this thesis, supplementary C++ code has been added for the linear PID controllers, feedback rolling-mean-value generations, feedback transport to the AC, and peripheral functions such as interference traffic generation, simulation parameter handling, and results handling. Furthermore, an algorithm for the packet controller, that is, to control the actual data streams partitioning between the ring directions according to the distribution controller, has been added. Figure 6.3 shows the UML class diagram of the added classes for an AC embedded in the ns-3 framework.

The dedicated AC simulation code for this performance evaluation consists of three classes derived from the ns-3 Application class, which is again derived from the Object

class. A detailed description of the ns-3 classes and their derivation chain is located at the ns-3 documentation home page https://www.nsnam.org/docs/. The functionality of the three added classes for the AC simulation is briefly described in Table 6.4.

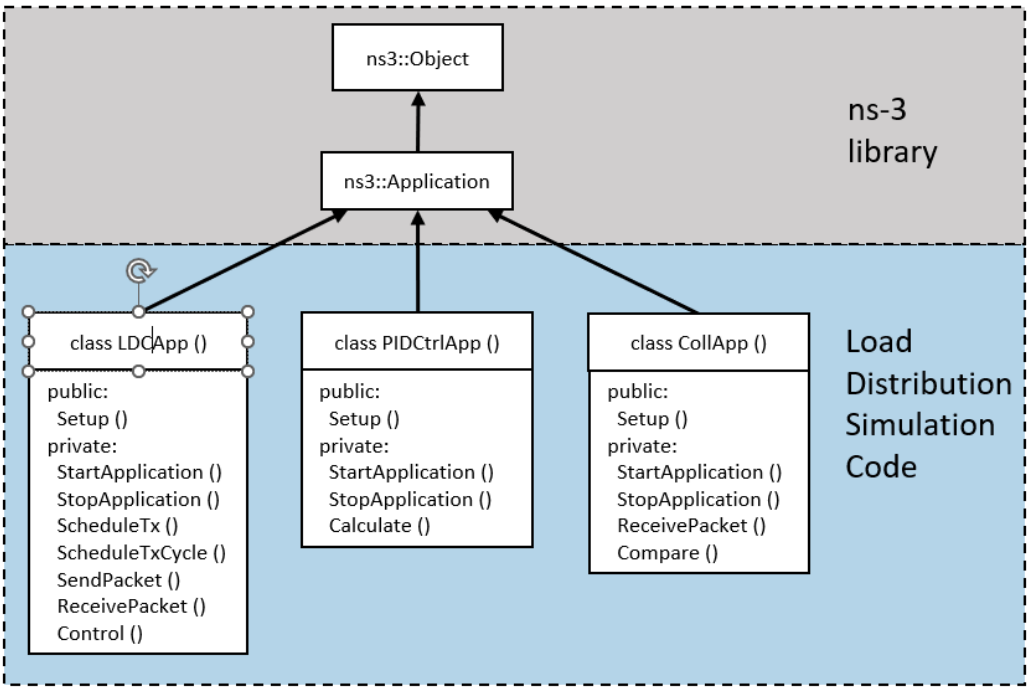


Figure 6.3: Dynamic Load Distribution Control Simulation with ns-3. Class diagram for Automation Controller

Table 6.4: Class Description of Simulation Code for an Automation Controller

Class	Purpose/Comments	Methods
LDCApp ()	Load Distribution Control Application. Entry application for packet handling. One instance for each ring direction.	StartApplication () Starts the Application, creates the packets, opens communication socket, and sends a first packet.
		StopApplication () Stops the Application, cancel send events, close the communication socket.
		SendPacket () Sends a data packet, schedules the next packet till burst end, calls scheduler for the next communication cycle start.
		ScheduleTx () Schedules the next packet to be sent.
		ScheduleTxCycle () Schedules the next communication cycle start.

Class	Purpose/Comments	Methods
		Control () Packet controller converts flow controller output to a change of number of packets to be sent. The pseudo code for this method is provided in Table 6.2.
PIDCtrlApp ()	PID controller class. Is used for distribution controller and for flow controller.	StartApplication () Starts the PID controller application, schedules first PID controller calculation.
		StopApplication () Stops the PID controller application.
		Calculate () Calculates PID controller and schedules the next calculation. The pseudo code for this method is provided in Table 5.3.
CollApp ()	Collector application. Is instantiated at automation controller. Receives the throughput feedback of the ring nodes and compares and provides the difference as input for the distribution controller.	StartApplication () Starts and initializes the collector application, schedules first collector call.
		StopApplication () Stops the collector application.
		ReceivePacket () Interrupt method for reception and classification of feedback frame.
		Compare () Builds the sum over the single application cycles per node and direction to build the overall throughput for application dedicated control. Finds the maximum of each direction and application cycle and also for all application cycles and builds and stores the difference of the directions. The pseudo code for this method is provided in Table 4.4.

The simulation code for a bridge or bridged end station is somewhat simpler because it only needs to provide the throughput measurement, including feedback sending. On certain nodes, an additional generation of interference traffic is added. Figure 6.4 shows the class structure of a bridge or bridged end station.

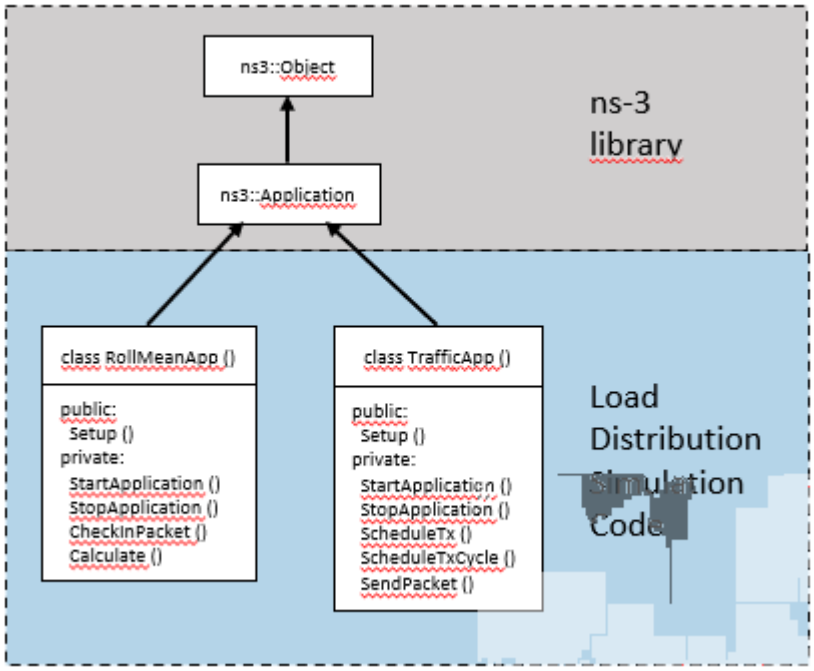


Figure 6.4: Dynamic Load Distribution Control Simulation with ns-3. Class diagram for a Bridge and Bridged End Station

The dedicated simulation code for a bridge or bridged end station for this performance validation consists of two classes. These are derived from the ns-3 Application class, which is again derived from the Object class as is the case for automation controller classes. The functionality of the two added classes is briefly described in Table 6.5.

Table 6.5: Class Description of Simulation Code for a Bridge or a Bridged End Station

Class	Purpose/Comments	Methods
RollMeanApp ()	Builds the rolling mean measurement of throughput on a node (bridged device) and per port. For each app cycle class, one RollMeanApp is instantiated as they finally will use different rolling mean measurement integration times.	StartApplication () Starts the Application, initiates first scheduling of the calculation method.
		StopApplication () Stops the Application, cancel feedback frame send events, close the communication socket for the feedback frames.
		CheckInPacket () Receive interrupt method to count amount, type and size of packets passing a port.
		Calculate () Calculates rolling mean for the node, port, and application cycle class. Possibility of calculation over all application cycle classes. Sends feedback frames to ACs. The pseudo code for this method is provided in Table 5.1.
TrafficApp ()	Sends frames in one direction of the ring, usually to simulate traffic interference not under control of the automation controller	StartApplication () Starts the Application, creates the packets, opens a communication socket, and sends a first packet.
		StopApplication () Stops the Application, cancel send events, close the communication socket.
		SendPacket () Sends a data packet, schedules the next packet till burst end, calls scheduler for the next communication cycle start.
		ScheduleTx () Schedules the next packet to be sent.
		ScheduleTxCycle () Schedules the next communication cycle start.

The detailed source code of the developed classes and methods for this simulation are provided in Appendix 2.

The **preconditions for the simulation** are as follows: In analogy to the flow control simulation in Section 5.6, streams of lengths of approximately 200 Bytes net SDU data load plus 42 Byte Ethernet header are assumed. For easier distinction in the diagrams, slightly different packet lengths, and thereby slightly different measured throughputs, were applied in the two directions in the ring. These approximately 200 Bytes lead to,

according to Equation (5-8), a frame transmission time of $T_{Tr} = 1.936\ \mu s \approx 2\ \mu s$. Therefore, to simulate a burst of CD streams, send events were scheduled every 2 μs. The EST window size was selected with $200\ \mu s$. Assuming a 1 ms network communication cycle, full use of the EST window is reached with the transmission of 100 streams and causes 100 percent CD load and thus a 20 percent overall bandwidth use by CD. The calculation of the distribution and flow controllers must be scheduled cyclically in parallel to their application cycles. The flow controller output influences the number of frames scheduled for each direction. At the nodes of interest, the feedback rolling mean of the bandwidth use, that is, the throughput value, is generated and fed back to the automation controller.

The simulation uses UDP data frames for both the transport of the throughput feedback values and for the CD frames. The frames contain information on their type of frame, either CD or feedback, application class affiliation, originating AC or node, and direction, which can be clockwise or counterclockwise.

Basically, the membership of a frame or packet to an application cycle class or an AC can be achieved by various network technology means, including:

- Grouping by stream destination multicast addresses
- VLAN memberships
- Dedicated identifiers for application cycle membership (e.g., application identifier, APPID) and automation controller membership (e.g., automation controller identifier, ACID) in the application frame payload. This is a common practice particularly in automation solutions such as the OPC UA PubSub protocol or PROFINET protocol.

For this simulation, dedicated APPIDs and ACIDs in the payload of the frames are used. TSN supporting switch hardware generally provides hardware facilities, such as Ternary Content Aware Memory (TCAM) filters, to analyse payload content at wire speed without additional CPU load or noteworthy delays.

The simulation begins with basic linear control without optimsation to form the basis for comparisons with the new optimised dedicated control method extensions. It is

then extended by applying application-cycle-dedicated distribution and flow controllers for the LDC.

The following subsections provide the results of the different use cases simulations.

6.7.2 Performance of the Basic Linear Control

The performance evaluation of the basic linear control method without optimisation is the first step in assessing the potential of the control method optimisation. For this purpose, a ring with a single automation controller hosting the load distribution control assembly is simulated. The simulated network ring is shown in Figure 6.2. An interference load μ_{I1} is introduced into the network, which provokes a step response by the load controller. The step response delivers load control quality results.

The network simulation has been set up under the following preconditions:

Use Case 1, Basic setup without load control:

- 1 Automation Controller (AC),
- 9 further bridged nodes,
- 1 application cycle of 1 ms,
- constant packet sizes of about 200 Byte payload,
- 200 µs EST window,
- 1 ms network cycle,
- The integration time for the rolling mean throughput measurement is equal to the application cycle time of 1 ms, as no other disturbances are to be expected in the network.
- bridge latency and LAN propagation delay sum up to 2 µs per hop.

As outlined in Chapter 5 , the 200 Byte payload result in an Ethernet frame of 242 Bytes, assuming data without a VLAN tag. With a data rate of 1 GBit/s, the 242 Bytes require a time of 1936 ns in the EST window. This implies approximately 100 frames per network cycle for a 100 percent CD bandwidth or 20 percent overall bandwidth use. The throughput calculation is repeated in a cycle time of 1 ms to guarantee a fast reaction to load changes. The basic load at the start of the simulation start is selected as 50 percent of the 20 percent for CD in each direction, clockwise and

counterclockwise. An interfering additional load of 20 percent is introduced at 20 percent simulation time at time $t_{i1} = 20\,ms$ from the interlink connection at node 3 in the clockwise direction. The target node for the interfering load is node 4. The interference is stopped at 60 percent of the simulation time, at $t_{i2} = 60\,ms$. This results in an interference duration of $t_{i12} = 40\,ms$ which is 40 percent of the simulation time of 100 ms.

Figure 6.5 shows the throughput μ(t) measurement in the clockwise and counterclockwise directions for use case 1 without any load control.

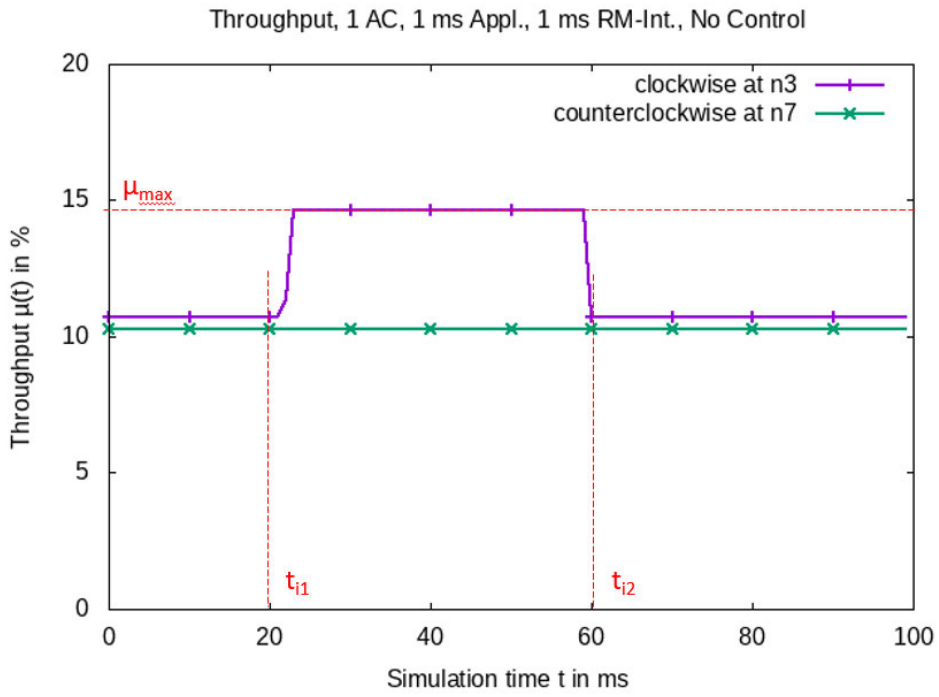


Figure 6.5: Use case 1: Throughputs over time without load control.

As shown in the diagram in Figure 6.5, the start of the interference at $t_{i1} = 20\,ms$ leads to a throughput μ(t) increase from 10.5 percent to approximately 14.5 percent in the clockwise direction, with no influence on the load in the counterclockwise direction.

Use Case 2: Basic setup with load control in the form of a basic proportional distribution controller and a PID flow controller:

The simulation network setup for use case 2 is identical to that for use case 1, with the difference that load control in the AC is enabled. The load control consists of a

basic distribution controller and PID flow controller. The full distribution PID controller is shortcut in this use case. The basic distribution controller calculates the difference between the clockwise direction throughput maximum and counterclockwise throughput maximum. The difference is then multiplied with the proportional factor of 0.5 and fed into the flow controller. The factor 0.5 takes the circumstance into account that the output of the flow controller feeds both packet controls for the two directions, the one for the clockwise direction and that for the counterclockwise direction, in parallel. Thus, packet control for one direction increases the output and the other decreases the output, each by half of the necessary load difference, to be balanced.

The flow control parameters for the PID controller to achieve the best control results are selected with a proportional factor $K_P = 0.4$, an integral factor $K_I = 160$, and a very moderate differential factor $K_D = 0.0002$. The integration time T_I results from the integral factor and the cycle time of the flow controller $T_c = 1\ ms$ and is thus calculated as $T_I = K_I * T_c = 160 * 1\ ms = 160\ ms$.

As in use case 1, the integration time for the rolling mean calculation was selected to be at the minimum of 1 ms as no slower periodic interference or other disturbances in the ring must be considered. This helps to determine the best possible control performance as a starting point for further simulations. Again, a 200 µs EST window is reserved for all CD, which is 20 percent of the network cycle of 1 ms; thus, 20 percent of the overall available bandwidth is also the maximally reachable throughput for CD.

Figure 6.6 shows the throughput measurement *$\mu(t)$* in the clockwise (purple) and counterclockwise (green) directions when the basic load control in the form of the basic distribution controller plus the flow controller is active. The gray dashed curves are the results of use case 1 above of load-uncontrolled operation for comparison.

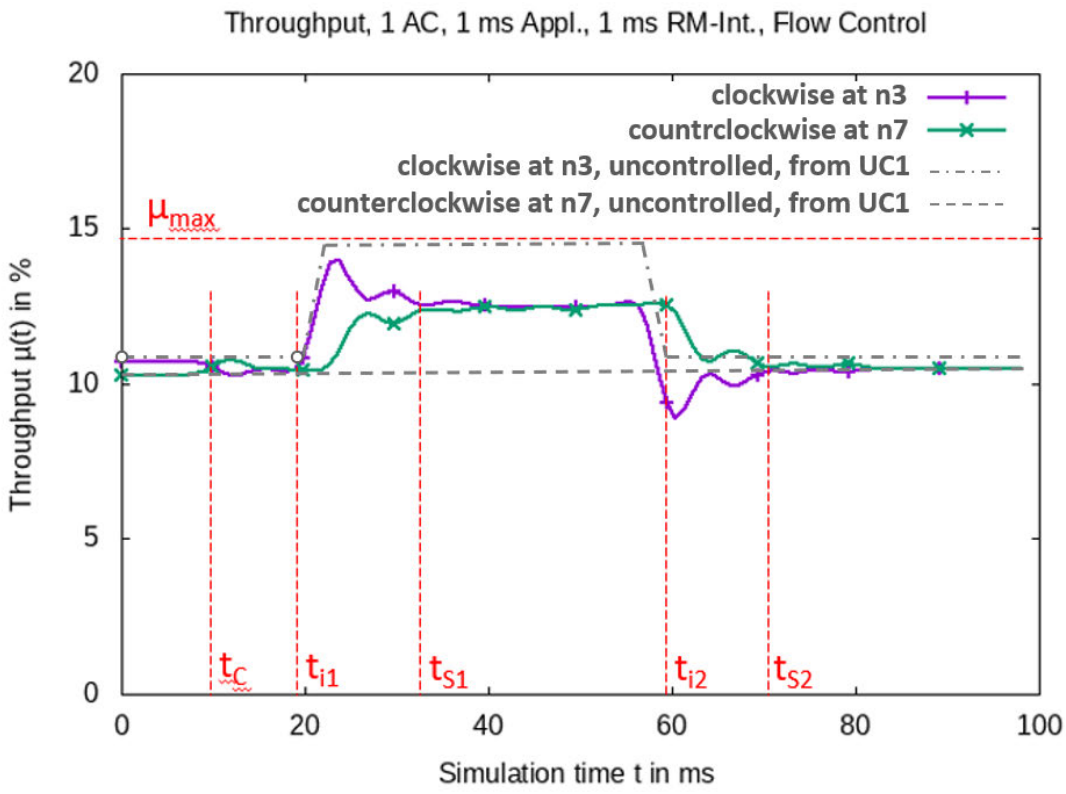


Figure 6.6: Use case 2: Throughput with basic load control.

From the start of the simulation to the control start $t_C = 10\ ms$ the throughputs are slightly different owing to the slightly different packet lengths to reach a better initial graphical differentiation of the two graphs. At $t_C = 10\ ms$, the load control is activated causing an alignment of both throughputs. After the clockwise interference load step at $t_{i1} = 20\ ms$ to approximately 14.5 percent, both directions load settle down to a common value of approximately 12,5 percent. This is achieved at the settling point $t_{S1} \approx 32\ ms$ within a period of approximately 15 to 20 ms settling time. At $t_{i2} = 60\ ms$ the interference load is removed, and 15 to 20 ms later at $t_{S2} \approx 70\ ms$ both the clockwise and counterclockwise throughput have settled on a common value of approximately 10 percent again. Thus, it can be stated that the basic load control achieves a step response settling time of $t_S < 20\ ms$ under the given preconditions and an acceptable maximal overshot of $\mu_O < 5\%$.

Use Case 3: Basic setup with full load control in the form of a PID distribution controller and a PID flow controller:

The simulation network setup for use case 3 is identical to the setup for use case 2 with the difference that the load control in the automation controller now consists of the full distribution PID controller and the PID flow controller. The empirical tuning of the cascade control leads to similar flow control parameters as in use case 2 and the following distribution PID controller parameters. These were determined as a proportional factor of $K_P = 0.2$, an integral factor of $K_I = 70$, and again a very moderate differential factor of $K_D = 0.0001$. The integration time T_I results from the integral factor and the cycle time of the flow controller $T_c = 1\,ms$ and is thus calculated as $T_I = K_I * T_c = 70 * 1\,ms = 70\,ms$.

Figure 6.6 shows the throughput measurement *μ(t)* in the clockwise (purple) and counterclockwise (green) directions when full load control in the form of the complete distribution PID controller plus the PID flow controller is active.

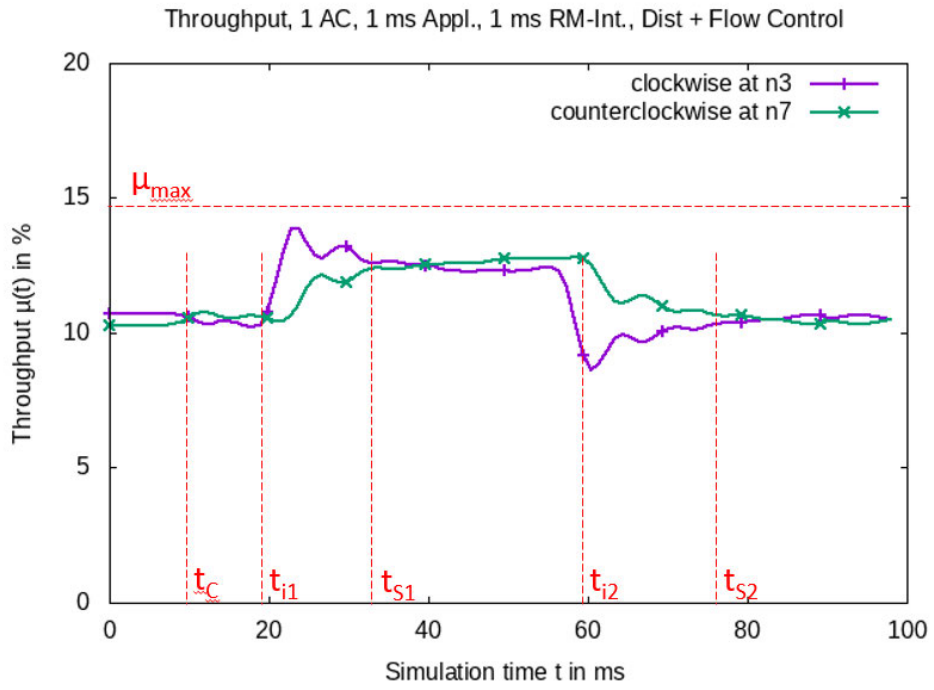


Figure 6.7: Use case 3: Throughput with full load control

As in use case 2, the load control takes effect at $t_C = 10\,ms$. With full load control, the load change step response at $t_{i1} = 20\,ms$ is compensated by the load controller at the settling point $t_{S1} \approx 32\,ms$ within a period of approximately 15 to 20 ms settling time. The settling point after the removal of the load change at $t_{i2} = 60\,ms$ is also reached at $t_{S2} \approx 75\,ms$. The slight difference in the two load-controlled throughputs

in the range of 50 ms to 60 ms results from the limited control resolution in steps of the number of data packets.

Summarising use cases 2 and 3, it can be concluded that the use of the full distribution controller cascaded together with the flow controller leads to a similar fast step response convergence as the basic distribution control of use case 2. The full distribution control displays a slightly wavier throughput output, which results from the two integration elements of the flow controller and distribution controller. They produce small oscillations at short rolling mean measurements of $1\ ms$. A settling time of $t_S < 20\ ms$ can be realised under the given preconditions with an acceptable maximal overshot of $\mu_O < 5\%$.

For the following use cases, the full load control setup consisting of the full distribution PID controller and PID flow controller is applied and hence forward named and referenced with the generic term “load controller” as a placeholder.

The next use cases contain additional applications on the AC sending the CD at additional, slower application cycles.

Use Case 4: Several applications with different application cycles and different throughput measurement rolling mean integration intervals:

This use case is provided simply to formally validate the statement that a common rolling mean measurement over all application classes traffic requires an integration time that is at least longer than the slowest application cycle time in the network domain. For this use case, three further applications are activated on the automation controller AC1. One application with an application cycle of $T_{App} = 2\ ms$, one with 4 ms, and one with an 8 ms application cycle interval. Each application starts sending in the first network cycle and sends 10 packets per network cycle. All of this CD traffic is measured using a common throughput measurement for all application cycle class intervals. The throughput is measured with different rolling mean integration intervals T_{int} as shown in Figure 6.8.

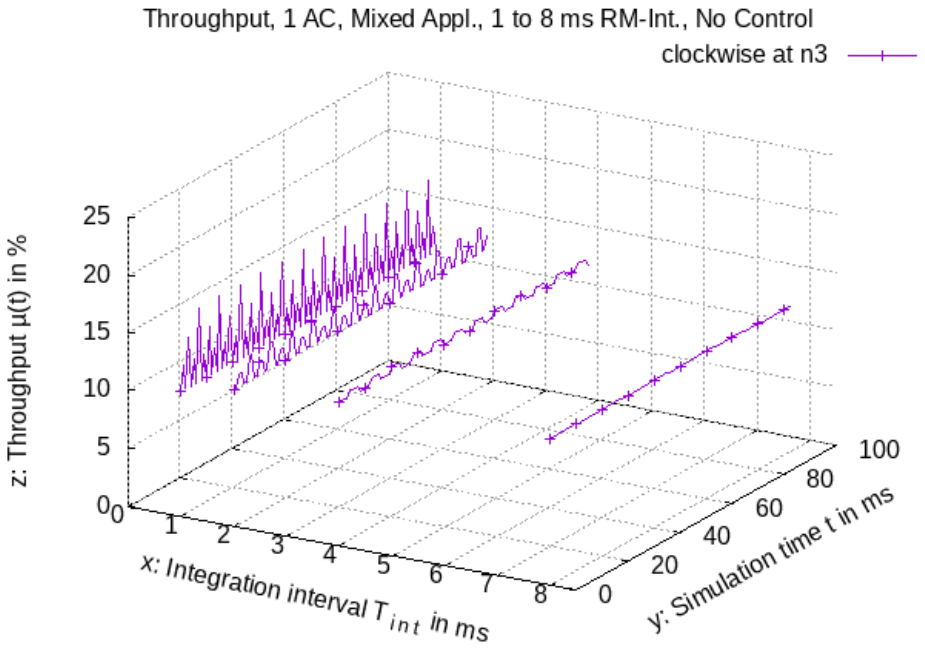


Figure 6.8: Use case 4: Throughput measurement over all application class cycles with different rolling mean integration intervals.

If the integration interval is kept at 1 ms as in use cases 1 to 3, the measurement result will be a wavy throughput actual value for the load control, as simulated in the very left plot at $T_{int} = 1\ ms$. With increasing integration interval T_{int} the ripple decreases. As expected, the ripple is only lost if $T_{int} >= T_{App}$ as simulated with the very right plot with $T_{int} = T_{AppSlowest} = 8\ ms$. The simulation thus confirms the obvious expectation that the rolling mean measurement integration interval must be similar or larger as the slowest applied application cycle interval in the automation network to avoid control oscillations. This is true if throughput measurement is performed over all application cycle classes for common load control.

Use Case 5: Fast application cycle traffic load changes in a common load measurement environment over all application cycle classes.

This simulation use case investigates the consequences for fast application cycle load changes if slower application cycle classes are introduced into the network. As with use case 4, CD traffic with an application cycle interval of $T_{App} = 1\ ms$, 2 ms, 4 ms, and 8 ms is applied at AC1. A common throughput measurement and a common load

control over all four application cycle classes is applied. Because of the results of use case 4, the rolling mean throughput measurement integration interval is set equal to the slowest application cycle interval of $T_{int} = T_{AppSlowest} = 8\ ms$. First, the use case is simulated as use case 5.1, without adapting the distribution controller and the flow control parameters that resulted from the optimisation for $T_{int} = T_{AppSlowest} = 1\ ms$ from use case 3. The results are shown in Figure 6.9.

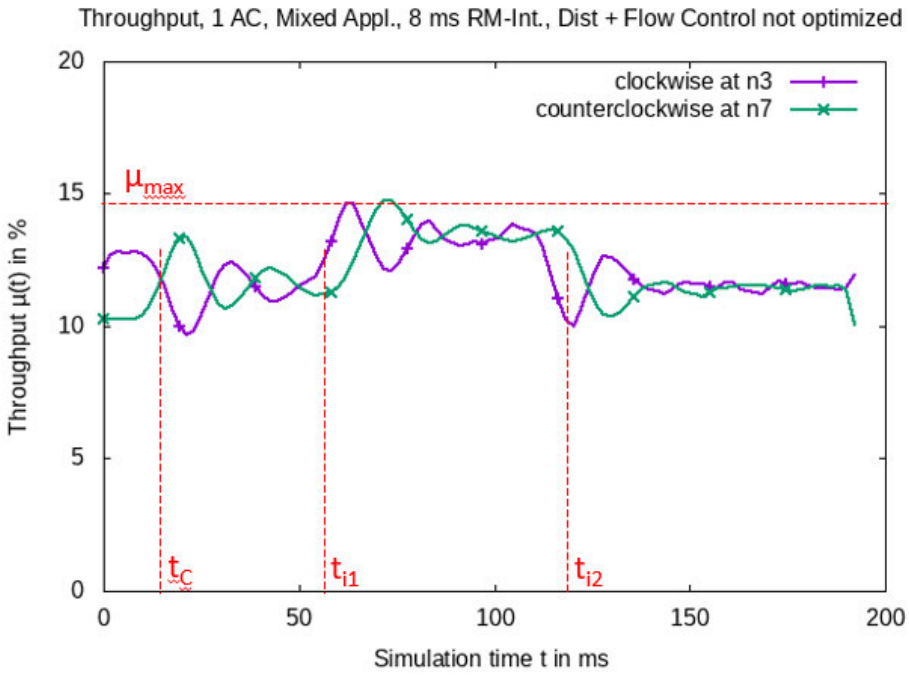


Figure 6.9: Use case 5.1: Fast application cycle CD load control deterioration under the influence of applications with slow application cycles and without load control adaptation to longer load measurement integration intervals.

As the simulation plot in Figure 6.9 immediately shows, the control result is a wavy throughput distribution. This results from the fact that the distribution controller and flow controller were not adapted to the changed control characteristics. These were caused by the changed PT1 characteristics of the rolling mean measurement in the control feedback. Figure 6.10 shows the simulation results of the optimised distribution controller and flow controller fitting to the longer rolling mean measurement integration interval.

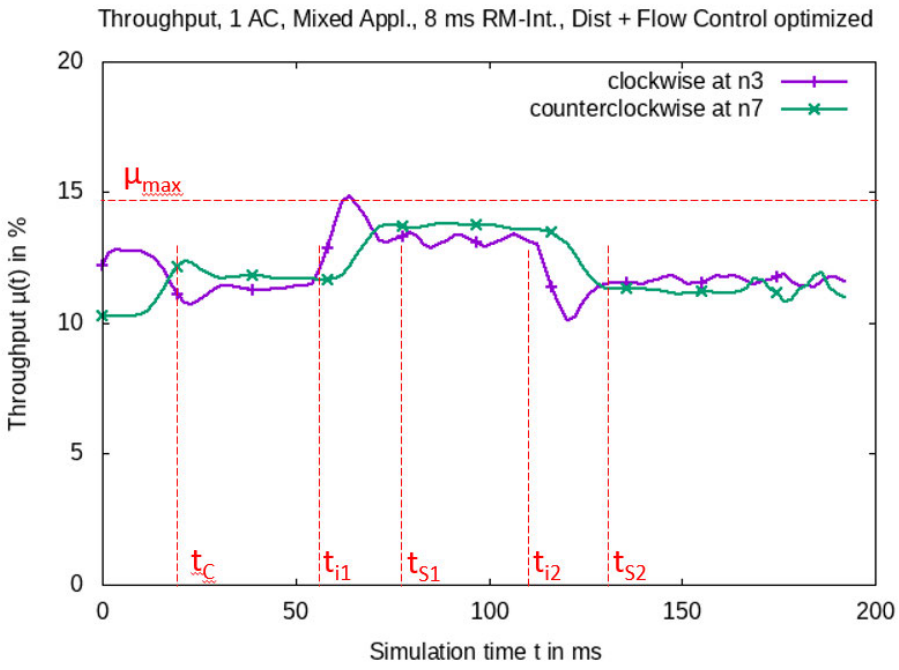


Figure 6.10: Use case 5.2: Fast application cycle CD load control deterioration under the influence of applications with slow application cycles and with load control adaptation to longer load measurement integration intervals.

The result of the controller parameter adaptation is a far lower ripple in the throughput distribution. Regarding the settling time, the comparison with use case 3 shows the expected deterioration from approximately 12 ms to 20 ms because of the longer rolling mean measurement integration time.

The conclusion from the simulations of use case 1 to 5 is that the 1 ms application cycle load changes can be compensated under the stated preconditions within a settling time of about 12 to 15 ms. The simulation results confirm the expectation that if slower application cycles are added to the network domain the integration time of a common throughput measurement must be increased until it is equal or bigger than the slowest application time. This is necessary to avoid the ripple of the throughputs to achieve satisfactory control results. However, this leads to the consequence that the achievable settling time decreases. In the case of the applied simulation use case, the achievable settling time for load changes deteriorates to approximately 20 ms.

With the next simulation steps, it is verified that the application cycle dedicated throughput measurement and application cycle dedicated throughput load control, as the main features of the proposed optimised control method for MAN, improve the overall throughput load distribution settling time.

6.7.3 Performance of Application Cycle Dedicated Load Control

To verify the control optimisation improvement of the proposed application cycle dedicated load control, two further use cases are simulated. Multiple application cycles and multiple interference traffic sources of different application cycles are added to the network. First, a common load control is simulated in use case 6. This is then compared with an application cycle dedicated load control in use cases 7.

Use Case 6: Several applications with different application cycles, combined with several interferences with different application cycles, and all controlled by one common load control:

Figure 6.11 shows the simulation setup from use case 4 with the four different applications hosted by AC1 sending CD μ_{AC1cw} and μ_{AC1ccw} at four different application cycles of 1 ms, 2 ms, 4 ms and 8 ms.

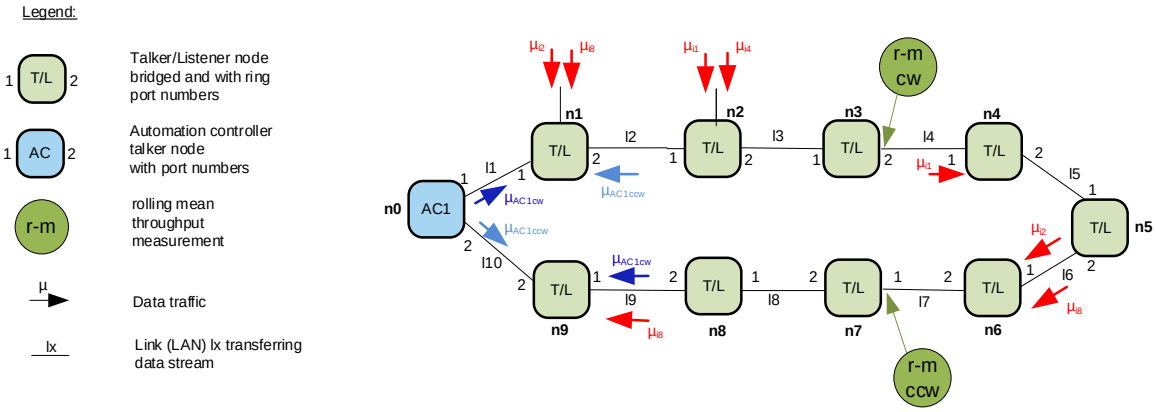


Figure 6.11: Use Case 6 setup

To evaluate the performance of a common load control, four different interference traffic loads μ_{i1}, μ_{i2}, μ_{i4}, and μ_{i8} are introduced into the network in the clockwise direction. They are also sent in four different application cycles of 1 ms, 2 ms, 4 *ms* and 8 ms. Interferences are added stepwise.

At node n2, the interference μ_{i1} with a 1 ms application cycle from use case 3, introduced at time $t_{i1} = 50\,ms$, is still active. Another interference with an application cycle of 4 ms is introduced at n2 at time $t_{i4} = 250\,ms$ bound for n9. At n1 an interference load with an application cycle of 2 ms at time $t_{i4} = 150\,ms$ and one with an application cycle of 8 ms at time $t_{i8} = 350\,ms$, both bound for n6, are introduced. To evaluate the control performance of the common load control, a throughput measurement with an integration time equal to the slowest application cycle time of 8 *ms* is applied. The necessary simulation time is extended to 400 *ms*. Each interference load creates a load step of approximately 2 percent of the overall bandwidth which is equal to approximately 10 percent of the available CD bandwidth. Figure 6.12 shows the load distribution without load control simulated as sub use case 6.1.

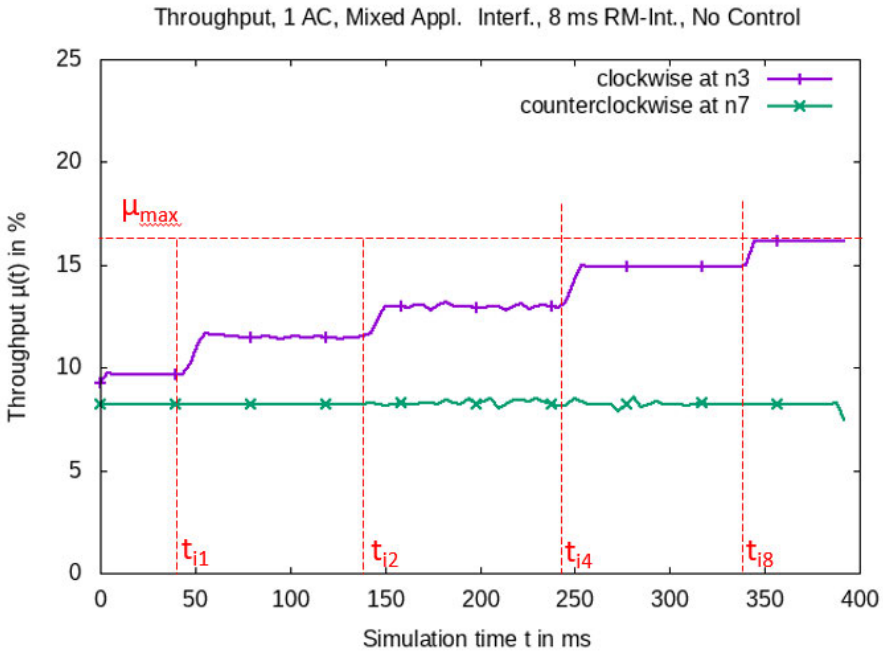


Figure 6.12: Use case 6.1: Load distribution for use case 6 without load control.

The longest application communication cycle is $T_{App\,max} = 8\,ms$. Thus, an $T_{Irm} = 8\,ms$ for the rolling mean measurement is the minimum integration interval to avoid oscillating load measurements. For the simulation of use case 6.2, the load control is

activated. Figure 6.13 shows the result of the load control in combination with $T_{Irm} = 8\ ms$ rolling mean integration interval.

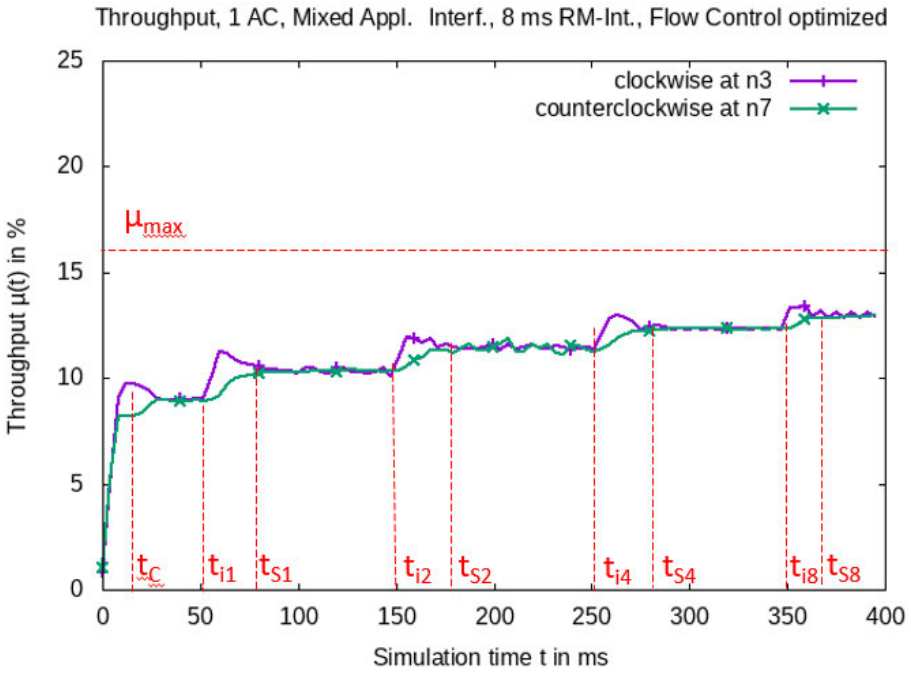


Figure 6.13: Use case 6.2: Load distribution for use case 6 with flow control and 8 ms rolling mean measurement integration interval.

The flow controller parameters for the optimum result are a proportional factor of $\boldsymbol{K_P = 0.4}$, an integral factor of $\boldsymbol{K_I = 75}$, and a differential factor of $\boldsymbol{K_D = 0.00029}$. The plot shows a slight ripple owing to small congestions in the queues as interference packets sometimes arrive at the output ports at the same time as other traffic packets. With a minimum rolling mean measurement integration interval, these small congestions become visible. With common load control for all application cycle classes the rolling mean is measured over all application cycle classes. Therefore, the packet controller has no possibility to differentiate between the application cycle classes for an optimal selective packet control to compensate for the traffic interferences of certain application cycle classes. A possible alternative is to control the packet flow of faster application cycles first. If the load cannot be compensated thereby, slower application cycle packet flow will be manipulated. As a sophisticated packet control mechanism is not the focus of this work, the fastest application cycle of 1 *ms* was selected to transmit sufficient data to compensate for all interference

traffic from all application cycle classes. As for the comparison of the two use cases, it is not important whether to work with full or basic distribution control. As there is no real advantage of the full PID distribution controller, the basic distribution control with only proportional influence is applied. Generally, PID controller tuning is a bit of a challenge, as an optimal control for a certain application interference compensation can cause a deterioration for another application cycle class interference. This is because a reference step for slower application cycle classes has a flatter ramp-up owing to the longer send intervals. Therefore, finding the optimal PID controller parameters for a selection of application cycle classes is always a compromise. This is a further disadvantage of a common load controller for all application cycle classes. The load control for use case 6.2 in Figure 6.13 starts at $\boldsymbol{t_C = 15\ ms}$. The simulation shows that, with an integration interval of $\boldsymbol{T_{Irm} = 8\ ms}$, all four load interference steps are compensated after a maximum settling time of approximately 25 ms at $\boldsymbol{t_{s1}}$, $\boldsymbol{t_{s2}}$, $\boldsymbol{t_{s4}}$, $\boldsymbol{and\ t_{s8}}$.

For the next use case, 6.3, it is anticipated that a new application with a slow application communication cycle time of $T_{App\,max} = 32\,ms$ is attached to the network. Thus, the rolling mean measurement integration interval must also be increased to $T_{Irm} = 32\,ms$. Figure 6.14 shows the load control results if the load controller parameters are not adapted to the new situation.

The simulation shows that the load changes can no longer be compensated for by the load control between the single-load interference steps. To achieve this, the load controller parameter for the flow controller for optimum control must be adapted to a proportional factor of $K_P = 0.6$, an integral factor $K_I = 48$, and a differential factor $K_D = 0$. The result is shown in use case 6.4 in Figure 6.15.

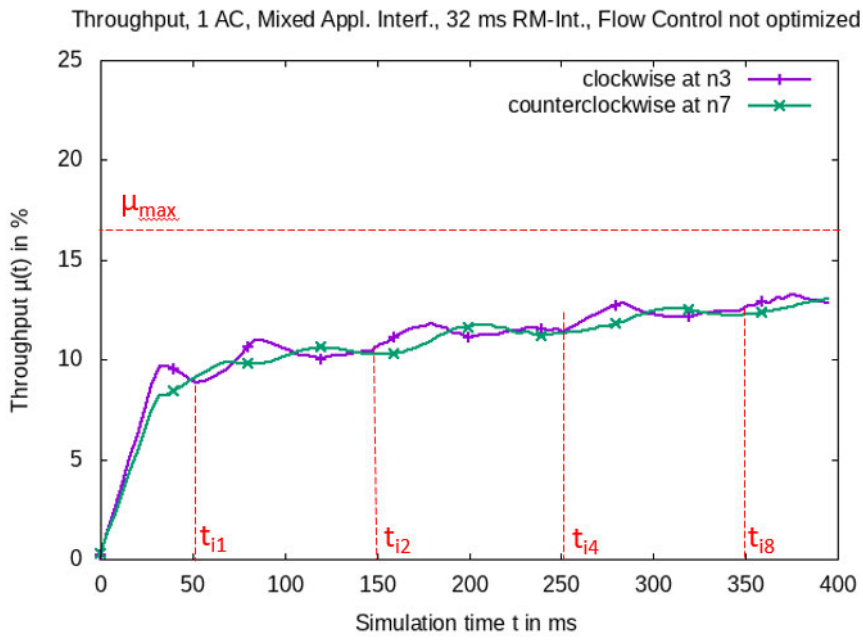


Figure 6.14: Use case 6.3: Load control results after connection of a slow application with application cycle $T_{App} = 32\ ms$ and adaptation of $T_{Irm} = 32\ ms$ but without adapting the load controller parameters.

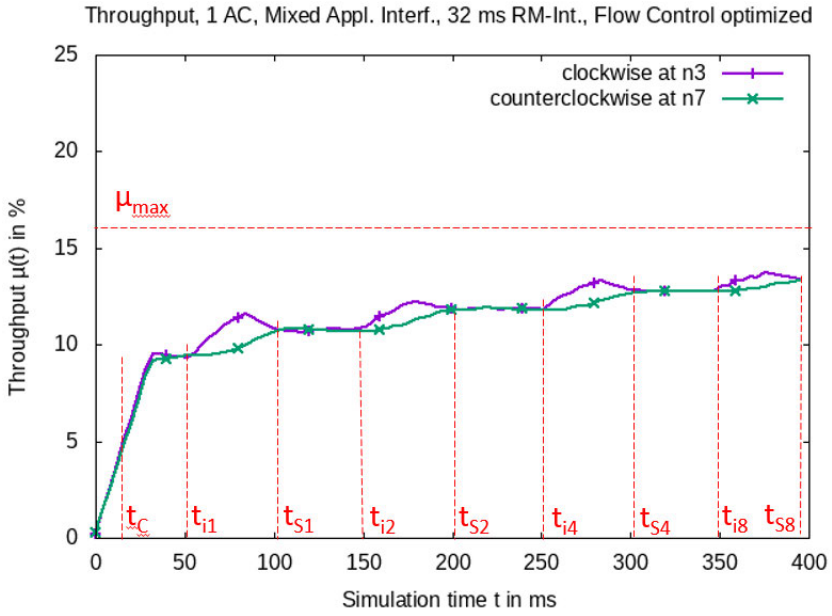


Figure 6.15: Use case 6.4: Load control results after connection of a slow application with application cycle $T_{App} = 32\ ms$ and adaptation of $T_{Irm} = 32\ ms$ and with optimised load controller parameters.

The simulation shows that the settling points t_{s1}, t_{s2}, t_{s4}, $and\ t_{s8}$ after the load interference steps at t_{i1}, t_{i2}, t_{i4}, $and\ t_{i8}$ are reached by far later, owing to the necessary longer rolling mean integration interval. The settling times after the interference steps increased from approximately 25 ms for $T_{Irm} = 8\,ms$ to approximately 50 ms for $T_{Irm} = 32\,ms$. The positive effect of the longer rolling mean measurement integration interval is a decrease in the waviness of the load when it is in a stable state after the settling time. Figure 6.16 compares the simulation outputs of use case 6.1 to 6.4.

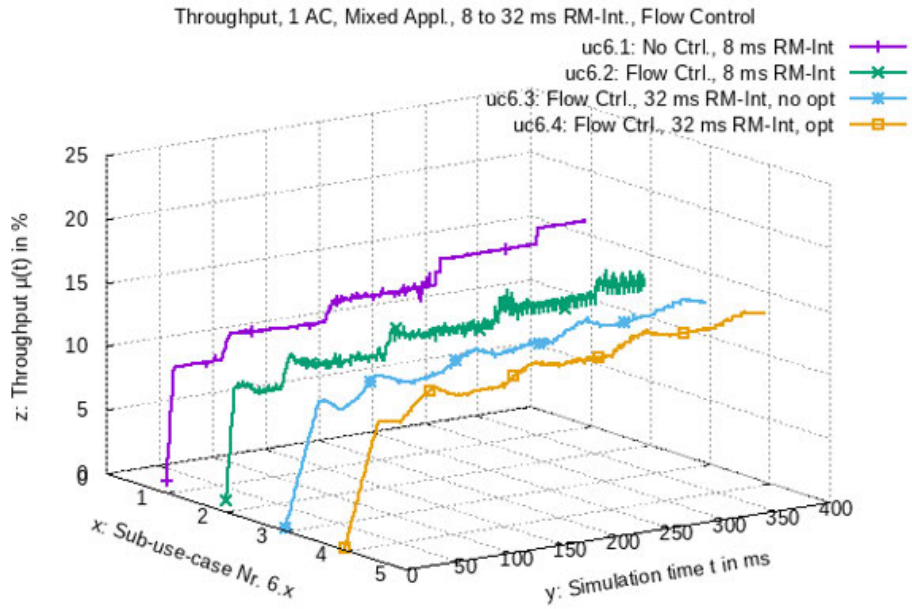


Figure 6.16: Use case 6 result: Comparison of the simulation outputs for clockwise direction.

The comparison shows that load control achieves a reduction in the maximum load from approximately 17 percent in use case 6.1 to approximately 13 percent in use cases 6.2 to 6.4. In use case 6.2, with the load control optimised for rolling mean measurement integration interval $T_{Irm} = 8\,ms$, the control reacts faster to interference load steps than with $T_{Irm} = 32\,ms$ in use case 6.3 and 6.4. However, the lower rolling mean measurement integration interval is also responsible for the higher load ripple. The plot for use case 6.3 (blue) with the unoptimised load controller shows higher load oscillations than the load in combination with the optimised load controller from use case 6.4 (brown).

In use case 6.5, the dependency of the settling time on the actual application cycle and the slowest application cycle is investigated. A number of slowest application cycle values, ranging from 1 to 32 ms are simulated.

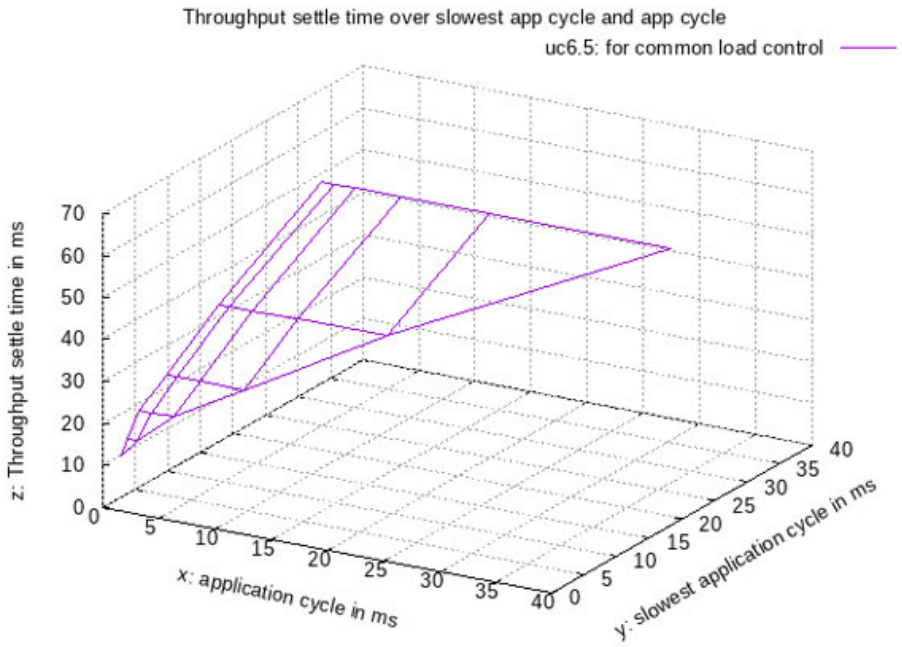


Figure 6.17: Use case 6.5: Load change settling time for common load control in dependency of application cycle and of slowest application cycle.

The plot indicates that the throughput settling time of certain application cycles depends only on the slowest application cycle time in the network domain. The settling times are constant over the application cycles for the discrete slowest application cycle times.

In the next step in use case 7, the control is switched to the application communication cycle dedicated load distribution control, which is the core improvement for the optimsed control method proposed in this thesis. This is achieved by adding load controllers to each application communication cycle.

Use Case 7: Several applications with different application cycles and separate load distribution controls:

For the simulation of use case 7, the same network setup as that of use case 6, as depicted in Figure 6.11, is used. However, the difference is that the load control within

AC1 implements independent load controllers for each application communication cycle of 1 *ms*, 2 *ms*, 4 *ms* and 8 *ms*.

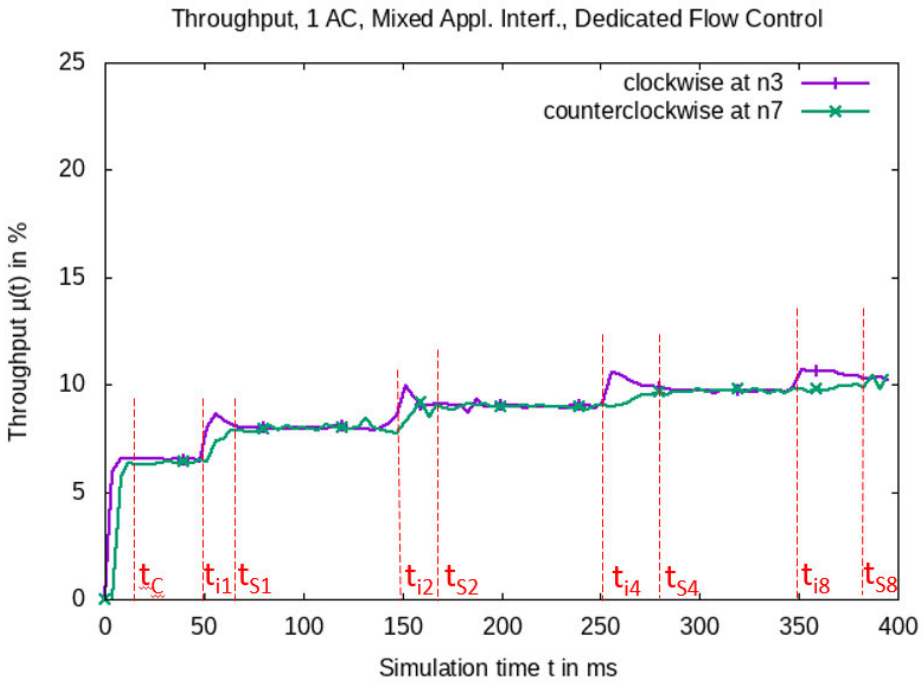


Figure 6.18: Use case 7: Load control results with application cycle dedicated load controllers.

The results for load control with application-cycle-dedicated load distribution controllers confirm the expectation that load changes of faster application cycles can be compensated for faster. Table 6.6 summarises the results. Figure 6.18 plots the sum of the load caused by all four application cycle classes of 1 ms, 2 ms, 4 ms, and 8 ms. Differently than in use case 6.2, displayed in Figure 6.13, which shows a constant maximum settling time of approximately 25 ms to 30 ms at $\boldsymbol{t_{s1}}$, $\boldsymbol{t_{s2}}$, $\boldsymbol{t_{s4}}$, $\boldsymbol{and}$ $\boldsymbol{t_{s8}}$, the faster application cycles settling times have now improved to $\boldsymbol{t_{S1}} \approx \boldsymbol{12\ ms}$, and $\boldsymbol{t_{S2}} \approx \boldsymbol{15\ ms}$.

Table 6.6: Settling times of common control and application-cycle-dedicated control for different application cycle times

Application Cycle (ms)	Common control settling time t_S (ms)	Application-cycle-dedicated control settling time t_S (ms)
1	25 to 30	12
2	25 to 30	15
4	25 to 30	25
8	25 to 30	25 to 30

The improvement is as expected lower with the 4 *ms* application cycle time with $\boldsymbol{t_{S4} \approx 25\,ms}$. The result for the 8 ms application cycle is identical to that in case 6.2, with $\boldsymbol{t_{S8} \approx 30\,ms}$ as the rolling mean integration intervals are identical. Figure 6.19 compares the throughput settling times over the application cycle time and over the slowest application cycle time in a three-dimensional diagram with the diagram from Figure 6.17 for the common load distribution control.

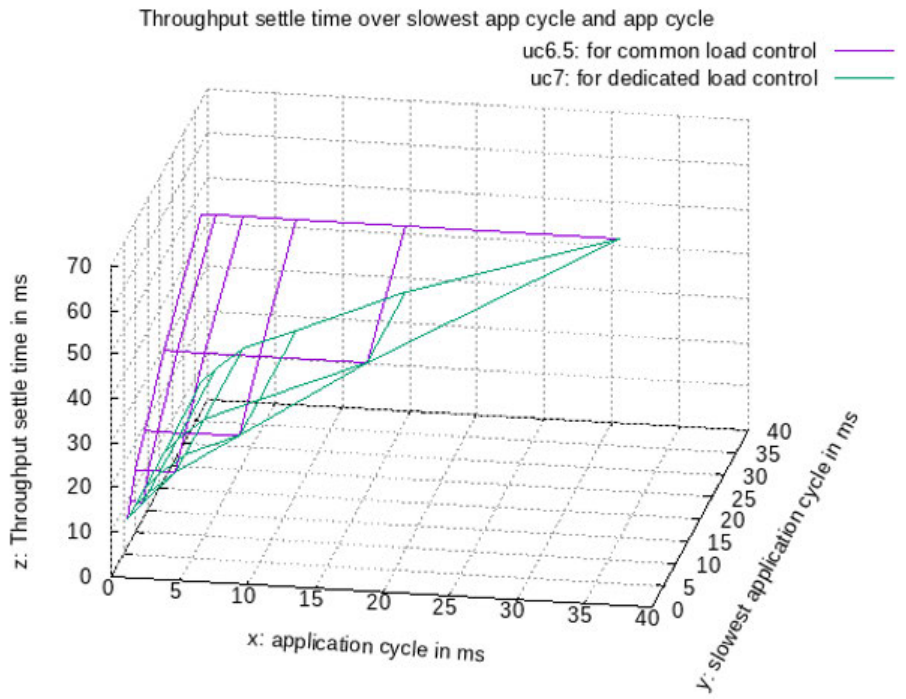


Figure 6.19: Load change settling time over application cycle and over slowest application cycle for application-cycle-dedicated load control.

Figure 6.19 shows that with the application-cycle-dedicated load control, the settling time (green curve) no longer depends on the slowest application but is constant over it. It now depends only on its own application cycle time. Thus, adding network participants with applications with slow application cycle times no longer influences the faster application load distribution convergence time.

6.7.4 Section Summary

The simulations clearly show the expected load control performance enhancement results. The classical and obvious load distribution control approach with common load control for all CD reveals the disadvantage that the control settling time of load changes depends on the slowest application cycle time in the network domain. This is due to the circumstance that the slowest application cycle determines the rolling mean measurement integration interval and thereby the inertia of the load control circuit. The simulation further shows that the new approach of using several application cycle dedicated load controllers, for the existing application cycle times prevalent in the network domain, improves the load control settling times for

applications with faster application cycles. Moreover, the load control of existing application cycle times becomes independent from the possibly newly added slower applications. In this case, the existing load distribution controllers must no longer be reconfigured, which is a crucial maintenance advantage when operating a MAN.

6.8 Network Error Mitigation Strategy

A further important question for network design is how to handle network errors in the form of link loss or a faulty node, as outlined in Section 4.9. In the automation network ring, this would cause an open ring and, consequently, only one available path from any talker to any listener instead of the two possible paths. For seamless traffic, this is, of course, no problem as data are sent doubly over both paths from the start. BE traffic is either also secured by the seamless protocol (HSR, PRP, or FRER) or covered by a path switch-over redundancy protocol such as MRP or RSTP/MSTP. This is also the case for non-seamless, non-stream CD, which is typically controlled by FDB-learning forwarding mechanisms instead of path control. Section 4.4 has outlined these mechanisms for different data-handling possibilities in MANs.

However, non-seamless streams that are under load control require a further mitigation strategy. From an optimum network use point of view, it would be ideal if all available paths and links were used near their maximum bandwidth capabilities. If this is the case, the loss of one link in the ring would have the following consequences:

1. Non-streams not secured by seamless transport from a lost path will additionally load the remaining path after the error occurrence and path switch-over by redundancy protocols (MRP or RSTP/MSTP in this case).
2. Streams not secured by seamless transport will also be shifted to the remaining path.

This would be problematic if the remaining path was already loaded near the maximum load before the switch-over. Possible solutions to this problem are as follows:

1. Maintain a reservation reserve for the load which could be subject to switch-over. However, calculating this load in advance can be quite a challenging task, as it

depends on the amount and distribution of traffic in the network domain. This requires constant bookkeeping and calculations during runtime.

2. Reducing the available bandwidth resources for data traffic of less importance in favour of more important traffic in the error case. With the EST, CQF and ATS, this can be achieved by extending the important gating windows while shortening less important traffic gating windows such as non-CD and BE. With SPQ, this is already an inherent system characteristic as non-CD and BE get lower priorities, and their traffic transport will be reduced in favour of additional higher priority traffic.

If resource reservation is applied, either distributed via MSRP or RAP, or centrally via a CNC, basically two reservation strategies can be applied, as outlined in more detail in Section 4.8.

1. Booking both paths with the complete traffic that would occur after the switch-over. This is especially recommended for slower resource reservation mechanisms such as MSRP.
2. Reserve the switch-over load bandwidth immediately after error occurrence. This is only recommended with faster resource reservation mechanisms such as LRP/RAP. The advantage of this method is that it is a simpler reservation process, as data does not need to be classified into the load actually transported on a path and the future load to be booked, as described in the previous bullet.

Which solutions are preferred depends on various factors such as whether a central or distributed network configuration has been selected, on the resource reservation protocol selection, and on the traffic classes in use. This must be decided by the system designer of the complete MAN.

The aspect of path control, although not the focus within this thesis, needs further attention to be paid by the network designer in case of the loss of link or device. The load-distribution control algorithm does not know which direction in the ring towards possible listeners has been lost. Therefore, it can be beneficial to switch from dedicated traffic-engineered paths within the TE-MSTI to FDB learning for the dedicated VLANs used by this traffic. Thus, the path to the listeners is automatically established by the MAC address learning process, as defined by IEEE 802.1Q (2022).

6.9 Chapter Summary

The most commonly used control methods for load control in communication networks for campus, mobile, and service provider networks were discussed in terms of their suitability for automation communication networks. In particular, the properties of the distribution core control method, flow control method, and possible feedback methods were considered. The classical PID controller was selected as a suitable representative core control method to satisfy the requirements of typical load control preconditions within TSN MANs. It is used for both the core distribution controller and flow controller to form the basis on which to build the optimisations.

The classic approach of a combined load measurement for CD for all application cycles has an important disadvantage in that the control inertia is determined by the slowest application cycle. The application-cycle-dedicated distribution control has been introduced as a major feature for an optimised control method. For this purpose, the throughput measurement at the paths output ports is also dedicated to the application cycle classes instead of a common measurement. For each node in the ring, an AC containing the load distribution controllers sets up a database with path delays and tailored controller parameters. This is required for an optimum control circuit configuration, wherever in the ring a load maximum is to be diminished. Furthermore, it is proposed to exclude slower application cycles from load control owing to their low load distribution contribution compared with the effort for a dedicated control effort. The smaller talker data traffic of smaller automation devices can also be excluded from the active distribution control to avoid the effort for controller implementation on such cost-sensitive devices. Alternatively, the traffic sending ring direction can be managed by the assigned AC`s LDCs.

Performance validations confirmed the expected improvements in the load distribution convergence time. Another important advantage is that new and slow applications added at runtime have no longer an influence on the established load distribution setup.

Chapter 7 Extension of the LDC Optimisation to Support Multiple Automation Controllers

7.1 Introduction

In Chapter 6 , an optimised load distribution control method for a single AC, such as a PLC or MC, in an automation network ring topology is introduced. The single AC per network ring use case is typical for a field-level ring as it appears in setups, such as single machines or smaller automation cells, as depicted in the lower part of Figure 1.1. Field-level rings are often connected to each other by redundant connections via a controller-level ring to communicate with each other. This use case is also referred to as the Machine-to-Machine (M2M) communication use case. An example is shown in Figure 7.1.

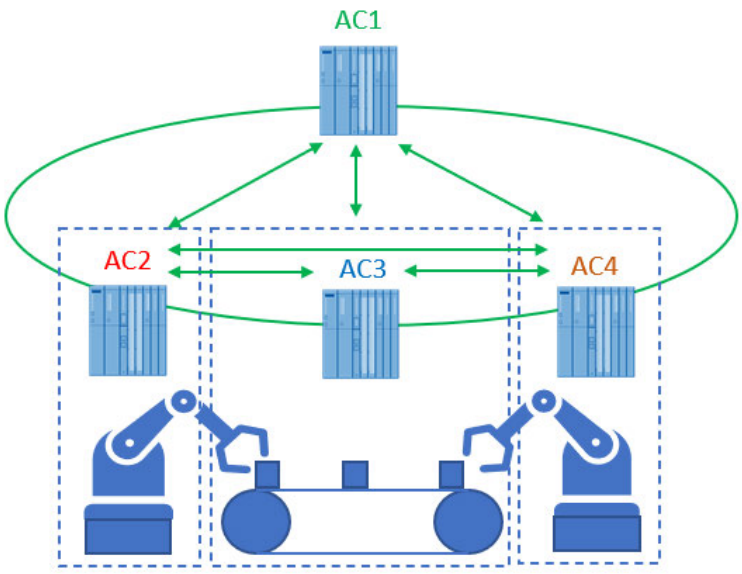


Figure 7.1: Machine to machine (M2M) communication

In this example, a supervisory AC1 communicates with the machines or automation cell`s AC2, AC3, and AC4. The machines/automation cells also exchange data. M2M or AC to AC is typically a 1:1 communication relation rather than a 1:n relation with many listeners. This means that there is usually only one listener in the ring subscribing to the streams of a certain talker. However, this has no effect on the load distribution. This is similar to considering streams for multiple listeners, where only the last listener

on a path is relevant for the extension of the stream into the network. For all the links in between, it is of no relevance whether the load is also consumed by listeners located nearer to the controller or if there is no more than one listener. Of course, there can also exist unidirectional CD, for example, from sensors to ACs or from ACs to actuators. The controller-level ring can also contain further ACs with higher-level automation tasks. These network structures result in the task of load distribution control involving multiple ACs at the same automation network ring.

The aim of this chapter is to extend the optimised load distribution control method from rings with a single AC from Chapter 6 to rings with multiple ACs.

7.2 Controller Location Considerations: Centrally or Distributed?

In Section 4.2, the basics and possibilities of a Central Load Distribution Control (CLDC) in comparison to a distributed approach (DLDC) were discussed. From an abstract point of view, a **central** intelligence controlling the network traffic of multiple ACs in an optimised manner would be an obvious solution promising best optimisation results. This central intelligence or Central Network Controller (CNC) can then contain algorithms to constantly calculate favourable traffic distributions at runtime. For example, Yan Song et al. (2021) use a combination of a modified Ant Colony Optimsation (ACO) algorithm and a K-nearest neighbor clustering algorithm to optimise a central routing and scheduling algorithm for industrial applications. Yang et al. (2021) also used an improved ACO algorithm for centrally calculating an overall schedule. Gavrilut et al. (2018) use a K-Shortest Path heuristic algorithm for routing and an Adaptive Search Procedure algorithm for scheduling.

The goal of these algorithms is not only an acceptable load distribution but also a minimum latency and no congestions. Another possibility for LDC is that the CNC contains an AI instance with a knowledge base on how to act under certain traffic conditions. A controller based on AI, or strictly spoken rather Machine Learning (ML) as the relevant discipline within AI, can contain a problem generator to generate stochastically distributed input vectors in the learning phase to provide the possibility for the controller to learn from its actions. However, the probability to be successful in preparing the ML controller with a complete set of all possible traffic constellations

of a real, possibly large plant with this method becomes lower, the bigger the networks. Therefore, the ML controller algorithm still learns some of the possible traffic constellations in the operating mode. However, such a learning phase during operation can also include erroneous or non-optimal behaviour on its way to finally achieve the best results. This is an undesired process within automation plants as there is no possibility of a learning phase in a real plant under real production conditions. To achieve optimum results from the beginning, the learning would have to be performed completely before operation via a model or a set of learning data providing basic start knowledge to avoid critical automation plant states, possibly causing damages to products, the plant, or even to humans. The better the environment perception of the ML agent, that is, the more precise the input data, the better the control results. Such input data would include for example source and destination addresses, application cycles, traffic classes, and frame sizes. Imagining a bigger network of hundreds of participants, which is not uncommon for manufacturing setups, this results in huge knowledge or action rules database. It would have to be built without having the real plant in operation. This could mean a very high input effort in advance to achieve a good approximation of the plant. Moreover, plant properties change with each additional or removed network participant or application on the host. These changes must be learned again before the system reacts appropriately. This is again a critical issue with automation networks, where the loss of control data must be avoided in any circumstance. Another disadvantage of a central solution is that it requires a central source of knowledge and action. A second hot-standby CNC is needed to achieve a redundant fail-save setup, which is an additional unwanted cost factor.

Distributed load distribution controllers avoid this disadvantage of a single point of failure. Each of the distributed load distribution controller can be responsible for a certain amount of load to be distributed in an optimised manner to achieve the common goal of an optimised load distribution. The failure of a single controller would only have an effect on a part of the load and not a complete loss of the overall optimisation function. However, one must realise that the result with distributed controllers cannot achieve the same distribution quality results of an "omniscient"

central solution. Despite this, because of the greater weight of the advantages, a distributed load distribution control approach is also the goal for the multiple AC problem in this thesis. A special aspect of a distributed solution is whether there is mutual influence of the controllers caused by common load paths. This can result in oscillating loads within the network. Whether this is the case is discussed in the following subsections.

7.3 Dependencies between Controller Instances

In the case of several automation controllers using the same automation network for communication with their devices, the use of a dynamic load distribution on more than one automation controller can create interdependencies between the individual load controllers. The reason is obvious. A change in the network traffic load on one path is recognized by different ACs load controllers that also use this path to communicate with their devices or other controllers. These will lead to the same conclusion to de-load this path. Therefore, the amount of shifted load is influenced by the number of controllers and could, without appropriate counter-measurements, disproportionately increase the load on the evasion path. This can lead to oscillations of the load or overload of paths. Thus, one challenge is to find a solution that compensates for these mutual dependencies.

Furthermore, the resolution of the dependencies is made more difficult because not all streams always have a common path. Streams from different talkers to one or more listeners somewhere in the ring may be forwarded on completely different paths in the ring, or they may only share a part of a common path. To provide a simple example of this problem, Figure 7.2 shows four ACs from which two pairs, AC1/AC2 and AC3/AC4, communicate with each other over paths P1 and P2.

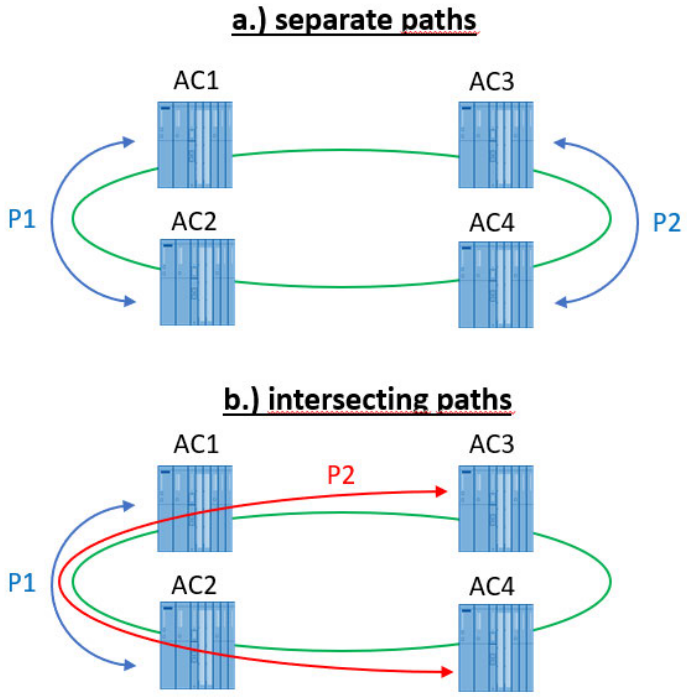


Figure 7.2: Consequences of path selection

In example a.), P2 between AC3/AC4 does not influence the path of AC1/AC2, whereas in example b.), when AC3/AC4 chooses the other direction in the ring, this is the case. Of course, with more complex controller network setups, and number of applications, these dependencies grow and can lead to communication dependencies that can be resolved only with unreasonably high effort.

These dependencies need either to be managed by an inter-working algorithm, or they need to be avoided from the start by a proper network-use design.

7.4 Discussion and Selection of Solutions

The expected mutual dependencies between multiple load distribution controllers at a ring can be counterbalanced in various ways. These reach from accepting and incorporating dependency into the load distribution control to complete decoupling of the controllers. With the latter solution, the dependencies do not have to be considered at all, at the expense of other disadvantages, as can be seen further down. The possibilities are closely connected to the applied Layer 2 TSN traffic shapers and

schedulers, as all of these have a certain influence. Thus, the first step is to clarify the influence of the shapers and schedulers on mutual controller dependency.

7.4.1 Influence of Traffic Shapers and Schedulers on Mutual Controller Dependency

The **CBS** is not suitable for automation applications because it distributes a burst of frames evenly over time. This would spoil the CD bursts with which the process data are transferred at the beginning of an application cycle, as outlined in Subsection 2.2.3. Therefore, this is not considered here.

One Layer 2 possibility to build an LDC for CD is to use the **SPQ** classical queuing method in combination with resource reservation. The latter is necessary to achieve deterministic behavior by excluding overload situations. A further requirement to reach deterministic behaviour is to assign this traffic class the highest QoS priority. Otherwise, frames would constantly be kept waiting in the output queue in favour of other higher-priority traffic spoiling determinism. As all ACs must be granted the possibility of using deterministic CD, they are all to be using the same high-priority traffic class. Thus, all the ACs CD traffic, which is subject to load control, is controlled via the same output queue in the bridges and bridged end stations. This creates varying latencies when the frames are transported through the network, which is unfavourable for the accurate design of applications.

Therefore, the SPQ is generally, regardless of its properties as to LDC, a rather unfavourable selection in terms of multiple ACs, if their applications depend on deterministic behaviour without much jitter. Although a bounded latency can be guaranteed owing to stream reservation, a certain determinism, that is, a guaranteed latency with a certain tolerance for jitter, is difficult to achieve. Regarding LDC, with the SPQ, all frame load changes measured by an ACs load control are also visible for other ACs and their load controllers thus creating a mutual dependency. Thus, the distribution control will always influence the CD of others ACs by choosing a specific path, unless other measures are taken. One such measure would be to provide a clear path when a frame is transmitted onto the network. This can be achieved by a central frame transmission algorithm that is aware of the complete traffic in the network.

Such a CLDC is in the position to calculate the exact transmission point in time for all or for certain important traffic of high priority in the complete network. If this is applied on at least the highest-priority CD, the jitter for the single frames can be kept at a minimum. However, a central solution is not the focus of this thesis, because of the disadvantages listed in 7.2. The SPQ thus creates mutual dependencies in combination with DLDCs.

EST allow the assignment of certain traffic to traffic classes. These can be assigned to dedicated gating windows in the network cycle as described in Subsections 2.2.3, 5.3.4, and Section 5.6. This fact can be used to separate traffic in a time-multiplexed manner. If each AC, with its internal load distribution controllers, is assigned to a different gating window, the LDC would also be decoupled. This approach is further discussed in Subsection 7.4.3.

The **CQF** traffic shaper typically works with smaller network cycle times, which are used to shift data frames from hop to hop in each network cycle, as explained and analysed in Subsections 2.2.3, 5.3.5, and 5.6. It also makes use of gating windows, similar to the EST, to reserve sending slots for traffic classes. Thus, in principle, it can also achieve the separation of traffic from different ACs in a time-multiplexed manner. The typical method for designing the CQF function is a one-buffer system, as described in IEEE 802.1Qch (2019). However, the traffic of several stream classes for several ACs cannot be separated into a single queue buffer. If each AC should be assigned a different stream class to achieve decoupling, different output queues must be emptied at different gating windows. This is why, at least with the classical CQF approach, working with a one-buffer system, there is also a mutual dependency between distribution load controllers residing on different ACs at the ring. One possibility to circumvent this would be to assign each controller different stream reservation classes combined with separate multiple CQF queuing systems. Multiple CQFs were also proposed by Finn (2019) for the separate handling of streams or stream classes. A two buffer CQF system as described by IEEE 802.1Q (2022), Annex T, can differentiate between two different stream classes, typically differentiated by priority, for two ACs without mutual influence. However, this requires an extension of the network cycle. The extension of the network cycle with the addition of further

ACs is also necessary with EST, but the EST method only needs one network cycle for all data traffic to traverse the complete network. In contrast, the CQF requires one network cycle per hop, which is a crucial disadvantage in terms of the implied higher dead time and the associated disadvantageous control behavior, as outlined in Chapter 5 . An alternative for a one queue buffer system to support further ACs including mutual independency would be the Per Stream Filtering and Policing (PSFP) feature belonging to the CQF shaper according to IEEE 802.1Qch (2019) . Using this, streams could be classified into Stream ID ranges, and not only Stream Classes differentiated via stream priority. Another aspect with multiple stream classes CQF is that it is a rather hardware resource-consuming requirement, as dedicated input filters in combination with buffers are required on each bridge. This is especially true if, for each stream class, a system of parallel input/output buffers for a better parallelism of the reception and sending phase shall be achieved, as proposed by Finn (2019). This could be problematic, as it is particularly for low-end bridges, not a matter of course to have these resources available. A clear advantage of the CQF is that it provides a rather easy to calculate delay, which depends mainly on hop count and network cycle time, as shown in Subsection 2.2.3 and Section 5.6. Thus, CQF always has the conflict of the need for small network cycles to obtain short end-to-end latency and the need for long network cycles to encompass all applications regardless of the length of their application cycle. However, in larger networks, this leads to long end-to-end delays which imply a need for a more sophisticated dead-time-compensating control design. Summarising these CQF properties it must be stated that the EST is better suited for a solution with mutual controller independency than the CQF.

In principle, the **ATS** is a combination of EST and asynchronously filled upstream buffers. Thus, it can also reach the separation of traffic in the same way as the EST, and also reach a mutual independency of load distribution controllers residing on different ACs. As outlined in Subsection 2.2.3, the difference is that the AST bridges are not synchronised. This means that the end-to-end latency from a controller to any maximum at a certain link is larger than with the EST, with the known disadvantages of the worse controllability of ATS LDC, as analysed in Section 4.6. Therefore, EST is

preferred over ATS, not only for the design of the optimal load distribution control solution for single AC, but also for a solution of mutual controller independency.

Preemption has no influence on the mutual dependency of ACs load distribution control because it only influences the latency of the highest priority stream class. This is then preemptive, as it can interrupt lower-priority frame transmissions in favour of its own stream class frame transmission on a port.

As a result of these considerations, Table 7.1 comprises the properties of the shapers and schedulers regarding their suitability to achieve mutual independency of load distribution controllers. It also contains strategies on how a distributed load distribution control solution based on the different shapers and schedulers can look like.

Table 7.1: Load controller dependency properties of traffic shapers and traffic schedulers and cooperation solution strategies.

Shaper/ Scheduler	Controller Independency possible?	Possible solution strategies for controller cooperation
SPQ	No	To cope with the mutual controller dependencies, an obvious strategy would be as follows: guarantee that only one controller at a time is influencing the traffic distribution. Furthermore, the single controllers should have only influence on own traffic sources and traffic from certain fixed assigned communication partners. However, this solution is expected to take more time than solutions with mutual independencies as it needs to work step by step instead of the parallel processing of the independent controls.
EST	Yes	EST supporting end stations, bridges, and bridged end stations are timely synchronised by PTP or gPTP. This fact can be used to reserve time slots within a network cycle for each AC and its load distribution controllers. Thus, the CD traffic of the different ACs can never influence each other if the ACs follow this assignment. On top of this, EST gating windows on output ports of the bridges can be used to secure that traffic reaching bridges outside the assigned time slots will be blocked by EST.

Shaper/ Scheduler	Controller Independency possible?	Possible solution strategies for controller cooperation
CQF	Yes	In the same way as with the EST, with CQF, end stations, bridges, and bridged end stations are time synchronised and the same solutions as with the EST can be applied. The difference is that the gating windows are applied between each node (hop) instead of one window to cross the complete path. Thus, the hop-by-hop cyclic forwarding involves a higher delay than EST, making the CQF rather a second-choice candidate in combination with multiple-AC LDC.
ATS	Yes	Considering the ATS in combination with multiple load controller cooperation, the crucial design feature is again the gating facilities at the output port. As with the EST and CQF, it can be used to separate the single ACs traffic classes to achieve a mutual independency. The single nodes are not timely synchronised as with EST and CQF. Therefore, an even higher and variable path delay must be accepted. This makes it much more difficult to apply flow control for higher bandwidth-consuming streams as discussed in Chapter 5 . These facts make the ATS a rather unattractive candidate for networks which should implement multiple-AC LDC.

The above discussions and the given solutions in the table show that the SPQ can be used for a solution that tolerates and solves load distribution controller interdependencies. It has the advantage of the best bandwidth use among all the four. From the three shapers and schedulers EST, CQF, and ATS, which allow mutual load distribution controller independency by design, EST is clearly preferred for its minimum path delay capabilities. These allow for the most efficient flow control setups as shown in Chapter 5 . The following subsections propose solutions for both the variants.

7.4.2 A Solution Including Mutual Controller Dependencies

If a traffic scheduler or shaper that implies load controller dependence, as is the case with the SPQ, needs to be selected as the hardware basis of the network domain for any reason, then a suitable method or algorithm to handle this dependence needs to be provided. This is hence forward named “Strict Priority Queuing based Distributed Load Distribution Control (SPQ-DLDC)” as a working title here for easier addressing.

As mentioned in the overview in Table 7.1, possible cooperation between the ACs can be achieved by timely decoupling of the ACs LDC process. Such a dependency compensation process, which must be in place at each controller in combination with the distributed approach, can be constructed as follows:

1. Every bridge or bridged end station in the ring, for example, an AC, distributes its traffic situation at its ring ports for each traffic class, application cycle class, AC assignment, and direction into the network. This can be achieved via multicast load updates, which are subsequently consumed by the nodes in the network. These build the throughputs classified by the application cycle according to Equation (6-1).

2. Every bridge or bridged end station, including all ACs, maintain a database that contains the traffic situation of the complete network.

3. One station acting as a load control administrator manages a temporal optimisation sovereignty token for the ring and leases this temporarily to the ACs, for example, in a round-robin system.

4. An AC receiving optimisation sovereignty checks whether it can improve the network load distribution either by maximum-reduction or optimum-distribution optimisation, depending on the selected optimisation strategy, as outlined in Section 4.6. The overall load at disposition for CD load control within the network domain is given by:

$$\boldsymbol{M}_{CD,ij} = \begin{bmatrix} \mu_{AC1,1} & \cdot & \cdot & \cdot & \mu_{AC1,\alpha_{max}} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \mu_{AC\gamma_{max},1} & \cdot & \cdot & \cdot & \mu_{AC\gamma_{max},\alpha_{max}} \end{bmatrix} \tag{7-1}$$

The rows are indexed by the controller assignment and the columns by the application cycle classes. The latter are required to apply the optimised application cycle class dedicated single AC control method from Chapter 6 on every AC in the network domain. The maximum-reduction optimisation goal from Equation (6-2) is the goal for each AC:

$$\min \max_{i,j \in \boldsymbol{V}} \mu_{ij,App\alpha} \text{ ; Subject to: } \forall\, e \in \boldsymbol{E}(G);\ \alpha \in \boldsymbol{A};\ AC\gamma \in \boldsymbol{AC} \tag{7-2}$$

The alternative optimum-distribution optimsation goal is changing in the same way from the form in Equations (4-5) and (4-6) to

$$\min \sum_{i,j=1}^{n} (\mu_{ij,App\alpha} - \mu_{M,App\alpha})^2 \tag{7-3}$$
$$i,j \in \boldsymbol{V}$$

Subject to:

$$\forall\, e \in \boldsymbol{E}(G); \alpha \in \boldsymbol{A};\ AC\gamma \in \boldsymbol{AC}$$

$$\mu_{M,App\alpha} = \frac{\sum_{i,j=1}^{n} \mu_{ij,App\alpha}}{2n} \ ;\ n \in \mathbb{N} \tag{7-4}$$

5. The optimisation process and core algorithm for an ACs LDC for one traffic class are shown in Figure 7.3. The solution described above can be classified into the category of dedicated algorithm controllers, as described in Subsection 2.3.3. This process cooperates with the load distribution control assembly setup consisting of the core PID distribution controller, PID flow controller, and packet controller, as described in Section 6.5. The maxima can be found by making a closed graph walk from each AC in each direction of the ring, where the AC represents the root of the graph. The optimisation algorithm enables the packet controller of the LDC circuit and feeds it with the Stream ID. This can then be used by the packet controller working as the flow controller output, thus achieving a reduction in the load difference for each application cycle class between the two paths originating from the AC currently being active. For a pure bridge or bridged end station not being an LDC container, the process shown in Figure 7.3 reduces to the first two tasks after initialisation, consisting of rolling mean building and distribution.

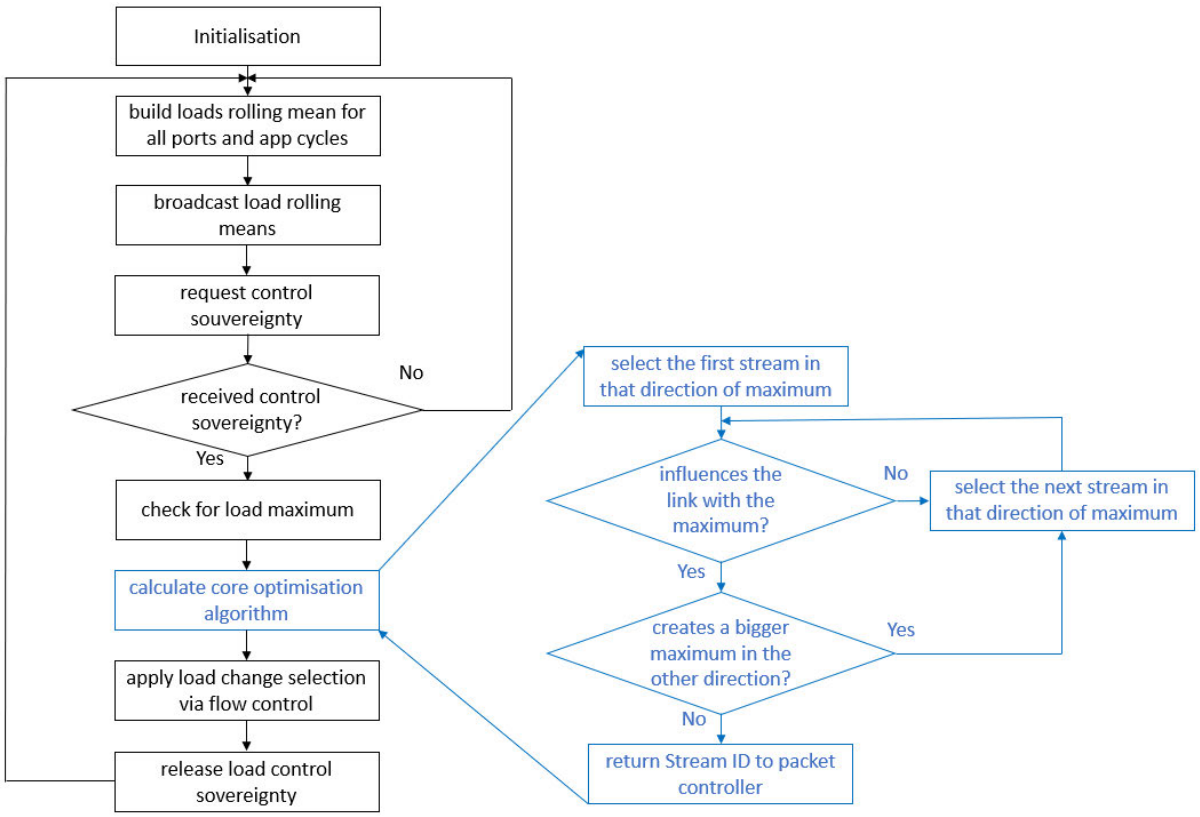


Figure 7.3: Dynamic load distribution control optimisation process and core algorithm

6. If the AC has finished this optimisation interval for all its application cycle classes, it releases sovereignty to the administrator station, which leases it to the next station.

7. The next AC then repeats this process until all ACs contributions reach a state without further improvements.

8. New exogenous traffic interference into the ring from higher-level or lower-level coupled networks, or from ACs with new applications, bring new load imbalances that lead to new optimisations.

   This optimisation strategy for finding better distributions towards a possibly optimum distribution can be classified as an “optimum-distribution” method, as described in Section 4.6. The described process demands that each AC is informed of which listeners in the network ring have subscribed to the CD streams it publishes. Moreover, each AC needs to know the distance (hop

count) to these listeners in each direction to assess whether the change of a stream or the division of a stream affects the maximum at the link in question. This also implies knowledge of the location of the maximum, and not only the direction. Such information is required to calculate what each change in direction would add to the other links load in the ring. With the described approach, an AC holding control sovereignty does not continue the optimisation until it has found an optimal solution, but rather applies the first found improvement. This ensures a fast application of this improvement instead of trying further possibilities with the risk of getting no better results, thus only wasting time. The alternative would be a more time-consuming calculation of all possibilities, leading to the advantage of finding the best result. Which of the two possibilities is advantageous in various situations can be the subject of further research, exceeding the focus of this thesis.

A clear advantage of selecting the SPQ as the hardware basis is that the available bandwidth is optimally used. This is because no gating windows are reserved for certain ACs, which might only be partially used. This would mean that the bandwidth lies fallow and is wasted and not usable for other ACs. However, this could be counterbalanced with a flexible gating window length for the EST-based hardware. The length could be adjusted after the load measurement as shown in the next subsection with EST-DLDC. With SPQ-DLDC, ACs can use whatever amount of the overall bandwidth they need without being mindful of gating window limitations. Of course, one could assign such bandwidth usage limitations to single ACs without the presence of hardware limitations such as gating windows. However, this would spoil the optimal bandwidth utilisation advantage of the SPQ-DLDC. On the other hand, this waiving of the individual resource reservation limitation also poses a risk. Traffic control mechanisms or reservation mechanisms will then only check the maximum available overall bandwidth corruption and not the individual ones for the single load controllers. One AC cannot increase its admissible maximum bandwidth to the overall available bandwidth at the cost of reducing the available bandwidth for others without influencing the others. However, the

optimum bandwidth use advantage could outweigh this. In this case, the most sensible bandwidth limitation border for resource reservation for each load distribution controller for the SPQ hardware basis in analogy to Section 4.8 is given as follows:

$$B_{SinglePathResMax} = B_{Max} \tag{7-5}$$

where $B_{SinglePathResMax}$ is the maximum reservation for single-path operation, and B_{Max} is the maximum overall bandwidth available, which is assumed to be 1 Gbit/s for the networks used here.

Moreover, the application cycle time also plays an important role in consideration of bandwidth use. With EST, long application cycles of multiple network cycles still make gating window reservations in every network cycle. However, these time slots are used only at the frequency of the application cycle. The result is worse bandwidth overall utilisation with solutions using gating windows. This is not the case with SPQ which is a further advantage of SPQ-DLDC.

Another advantage is that no time synchronisation protocol is required for frame transmission within the assigned time slot. This also enables less-expensive end stations to participate in the load-controlled network. This is often a crucial requirement for creating a MAN solution.

Regarding the detailed design of one ACs load distribution control for SPQ-DLDC, the same principle of separated controllers for each or for the most important application classes, which are differentiated by their application cycle, applies, as outlined previously. With SPQ-DLDC, the application cycle of a certain CD has a further influence. As outlined in Section 5.2, the rolling mean measurement time interval must be longer than the CDs application cycle to avoid oscillation of the measured value. This also means that load distribution controllers residing on other ACs will measure an applied change to its full extent only after the rolling mean measurement time interval has passed. Therefore, an AC must wait for at least:

$$T_{PassToken} \geq T_{Mean} \quad (7\text{-}6)$$

before passing the control sovereignty token back to administration and thus further to the next AC. $T_{PassToken}$ is the token passing time interval to wait, and T_{Mean} is the rolling mean measurement time interval from Section 5.2, of the application cycle class of the CD for which a path change is currently ongoing. This ensures decoupling and prevents other controllers from reacting to traffic situations that are about to change. Otherwise, the load changes would oscillate.

It is clear from these considerations, that the effort to compensate for the mutual dependency of load distribution controllers is extensive. The alternative is to exclude mutual dependencies from the start as discussed in the following subsection.

7.4.3 A Solution Avoiding Mutual Controller Dependencies

Summarising the discussion of the properties of the shapers and schedulers from Subsection 7.4.1, it can be stated that EST, CQF, and ATS are suited to achieve decoupled multiple ACs load control in combination with DLDC. Let us name these solutions **EST-DLDC**, **CQF-DLDC,** and **ATS-DLDC** respectively, in analogy to the SPQ-DLDC of Subsection 7.4.2 for easier addressing. The decoupling is achieved mainly by the reservation of time slots for the single AC's CD traffic within a network cycle. It must be stressed that this network global time-slot reservation is applicable for both I-CD and NI-CD. This is the case even though the latter is not sent synchronously to a network global working clock. It is only sent cyclically with the cycles derived from a device-local clock. However, NI-CD will be synchronised into the timing slots at the first bridge, that is, the edge bridge. The bridges themselves must be synchronised. The use of NI-CD allows for simpler and economical device design, saving cost sensitive devices the often tricky and demanding time synchronisation protocol support. The backside of the NI-CD is a higher latency as this waiting time at

the edge bridge for synchronisation to the time slots is added to the overall path delay.

Theoretically, this reservation of the time slot for sending in combination with a time-synchronised transmission of frames by ACs or bridges would be sufficient to achieve decoupling. However, a further measurement to exclude possible timing slot corruptions by ACs is to use a reserved time-controlled gate opening, that is, the gating window, at the output port. The gating windows are assigned to the different traffic queues of the traffic classes, one for each AC. This can be achieved using firmware for all three EST, CQF, and ATS schedulers in the TSN supporting bridge switch ASICs. As we have seen in the previous subsection, the SPQ shaper is not suitable for achieving this decoupled stream handling in combination with DLDC. This is because even scheduled transmission of frames does not guarantee uninfluenced transportation through the network. As outlined in the previous section, decoupling can be achieved only by applying control time slots instead of send time slots. Another crucial advantage of the EST-DLDC, CQF-DLDC, and ATS-DLDC is that separate queues are used for the gating windows. In contrast, with SPQ-DLDC, the streams of all ACs share the same queues in the bridges, implying that traffic interference arriving at any point in time thus influences the available load and the measured load for others. However, as already outlined in the previous section, in contrast to EST-DLDC, CQF-DLDC, and ATS-DLDC, SPQ and SPQ-DLDC manage without a time synchronisation protocol.

Thus, EST, CQF, and ATS traffic shapers are suited to achieve decoupling of the load distribution controllers working at the same automation network ring. This thesis focuses on the EST-DLDC solution because of its very low latency advantage as described in Chapter 5 . However, any conclusions drawn for EST-DLDC regarding load control decoupling apply equally to CQF-DLDC and ATS-DLDC solutions.

As stated above, decoupling is achieved by reserving dedicated stream classes and gating windows for each AC. Thereby, a time-multiplexed use of the

network is possible with each AC using a different time slot. This demands a time-synchronised sending of the I-CD or alternatively a cyclic sending of the NI-CD within each AC, and thus an application of a time synchronisation protocol such as IEEE 1588 (2019) or IEEE 802.1AS (2020) for the end stations (only for I-CD), bridges, and bridged end stations. Figure 7.4 depicts the solution.

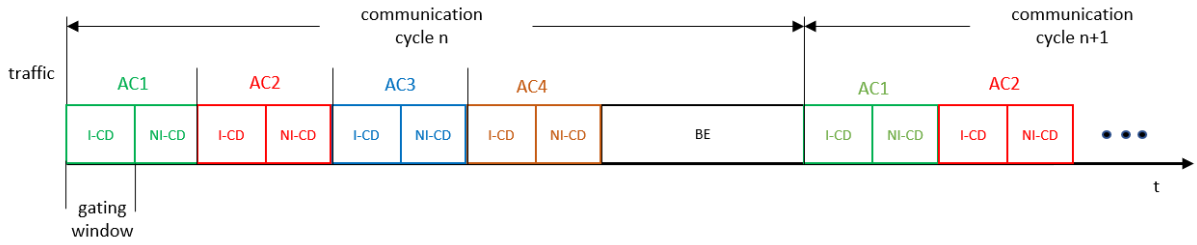


Figure 7.4: EST-DLDC solution for Automation Controller decoupling

In the example in Figure 7.4, the I-CD and NI-CD gating window slots are reserved for four ACs, AC1 to AC4, located in a ring, as shown in Figure 7.2. Locating the NI-CD slot of an AC directly after the I-CD slot of the same AC has the advantage that the gate for the I-CD of that controller can stay open during the opening of the NI-CD gate. Thereby, possibly late frames of I-CD are not blocked, although this should theoretically not occur if the network has been designed properly and the resource reservations have not been exceeded. After all CD gating windows of a network cycle follows a common Best Effort (BE) traffic phase within that cycle for all non-CD traffic. The common BE phase instead of single BE phases after the CD gating windows has the advantage that large frames can also be transported without being constantly interrupted or queued in favor of the next I-CD phase of the next AC. An SPQ can be applied within the BE window. As with the mutual dependency solution SPQ-DLDC from Subsection 7.4.2, each AC in the EST-DLDC has separated load distribution controllers for each or for the most important application classes categorised by their application cycle membership. The difference to SPQ-DLDC is the crucial parameter of the maximum admissible bandwidth. The decoupling is achieved by limiting the bandwidth maximum by the relation of the length of the gating window to the network cycle length:

$$B_{SinglePathResMax} = B_{Max} \cdot \frac{T_{GW}}{T_{Netw}} \quad (7\text{-}7)$$

where $B_{SinglePathResMax}$ is the maximum reservation for single-path operation, B_{Max} is the maximum overall bandwidth for the path, T_{GW} is the gating window length of the stream class, and T_{Netw} is the network cycle length. The lengths of the gating windows do not have to be fixed. They can be estimated at the time of engineering to meet the needs of individual ACs. Furthermore, a dynamic adaptation of the length could be applied in such a way that, for example, only partially used windows could be reduced or constantly full windows could be extended after such situations have been measured over a longer period. Figure 7.5 shows an illustration of an example of bandwidth use with the EST-DLDC.

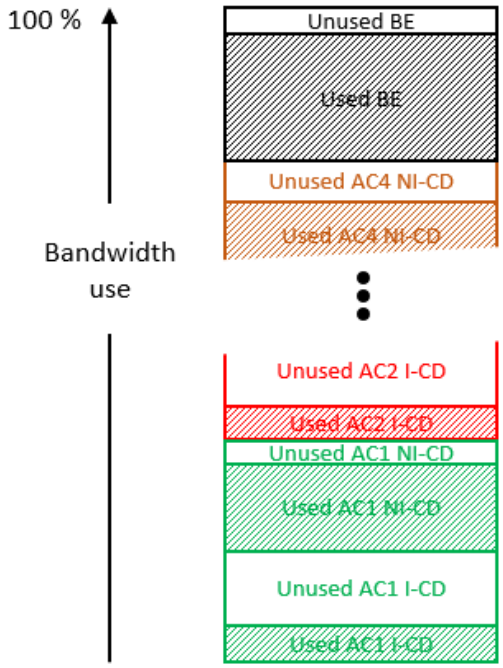


Figure 7.5: Illustration of EST-DLDC bandwidth use example

In this example, only AC1 NI-CD and BE show proper utilisation of bandwidth because the unused bandwidth is small compared to the bandwidth used. This means that the gating window for AC1 NI-CD is slightly longer than that required to have some reserve bandwidth. The AC1 I-CD window length is not ideal and could be reduced. This could

also lead to a smaller network cycle length if it fits the end stations' capabilities and needs. With respect to the bandwidth measurement interval, also with EST-DLDC, the bandwidth measurement interval must be a multiple of the longest application cycle class time $T_{App\alpha}$ as outlined in Section 6.5.

If a certain transmission and gating window is reserved for only one ACs CD traffic class, then one could justifiably ask whether there can be any traffic interference at all in the ring within that traffic class. One could further argue whether then a DLDC is necessary at all. However, the answer to this question is still yes. One advantage, even without any interference, is that the DLDC automatically cares for a smooth load distribution from the start, saving the network administrator 's effort into precise traffic engineering. In addition, unidirectional traffic to actuators or from sensors creates imbalances in the paths. This could happen within a stream class of an AC if this unidirectional traffic is assigned to this stream class, or it could be assigned to a reserved class. A common stream class is expected to be better suited. There are two reasons for this. First, the number of stream classes should be kept low for better bandwidth utilisation, as described by the incomplete use of gating windows above. Second, there are limited gating window configuration possibilities in the switch hardware. A further cause for load imbalances is that also seamless traffic, that is, traffic sent doubly in both directions, issued by ACs using the FRER, PRP, or HSR media redundancy protocols do not automatically leave a balanced network load distribution, as one might assume at first. This is evident in the example shown in Figure 7.6.

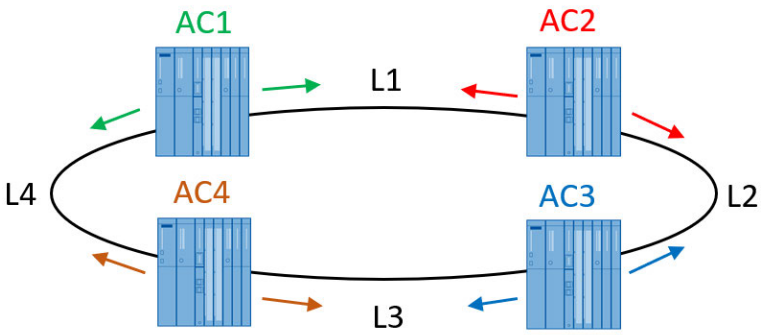


Figure 7.6: Automation setup with seamless traffic

Let AC1 be a head or supervisory AC communicating seamlessly with three field level ACs, AC2 to AC4, each of which controls, for example, a machine or a smaller automation cell. AC2, AC3, and AC4 also communicate with each other (machine-to-machine, M2M). All communication is assumed to be bidirectional, that is, the AC1 delivers, for example, control setpoints or other output settings, and the communication partner answers with control actual values or other input data. If we assume that the communication of AC1 with the other ACs consumes a 5 percent load in both directions and M2M consumes a 2 percent bidirectional load, we obtain a special case of homogenous load distribution over links L1 to L4 of 21 percent (see Annex 1 for detailed calculations). However, if we only had an asymmetrical communication load of AC1 to AC2, AC3, and AC4, assuming 5 percent from AC1 to others and 3 percent in the other direction, the load distribution is already inhomogeneous and justifies LDC. Figure 7.7 graphically shows the load distribution in the latter case. For detailed calculations, refer to Appendix 1.

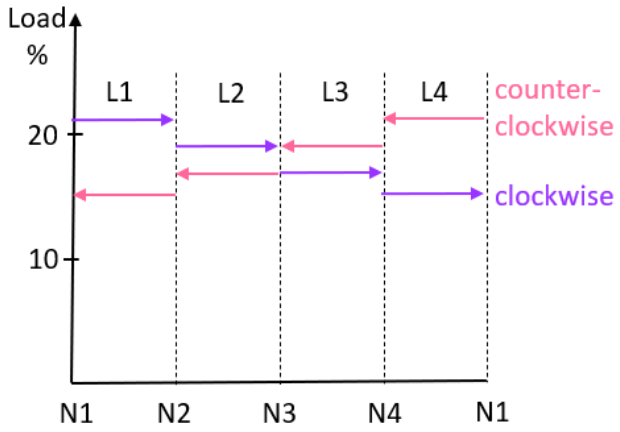


Figure 7.7: Example of load distribution of seamless communication with unsymmetric load.

If we assume further unidirectional sensor/actuator CD streams without LDC, the load distribution can become further inhomogeneous. Therefore, non-seamless CD traffic in the ring makes sense to be used as a means to achieve a more homogeneous load distribution. For example, let us assume further non-seamless traffic between AC1 and AC2 in both directions. In the direction from AC1 to AC2, both directions have the same maximum of 21 percent already, and it would therefore be best to divide the additional load on both paths. In the direction from AC2 to AC1, it would be best to send the traffic counterclockwise over L1. This is the better selection, as it would help both to avoid creating a higher value over L2 and using the shortest path. In addition, the shortest path achieves a minimum delay on top of the favourable load distribution. Thus, the LDC with its integrated packet controller provides the best path selection for this non-seamless traffic. However, even if seamless traffic is distributed homogeneously, additional uncoordinated non-seamless traffic spoils the distribution without an LDC. Again, the load distribution control algorithm ensures that all single-path traffic will be split automatically between the two possible paths, thus avoiding the local maximum to be on either of the two paths nearest the controller. From a practical point of view, non-seamless traffic can be marked either as minimum-delay traffic or as load-distributable traffic. A packet controller in a load distribution controller can assign the

shortest path to the traffic with minimum-delay requirement and use load-distributable traffic for load distribution optimisation purposes.

Another traffic type to be considered with LDC is bidirectional traffic to smaller end stations that do not provide their own LDC, but are under control of an AC. For example, this could be a drive for motor control. The AC, with its load distribution controller, can choose the path towards the drive. For the CD traffic in the other direction, from drive to AC, such as actual values of the motor current, speed, or position values, the drive could select the reception port of the setpoints to be also the output port for actual values. Another solution can be that the AC manages the drive output port usage. This can be achieved by either a flag in the header of the CD or by a generally manageable object in an end station, the drive in this case, at the control plane level. Thus, the AC can influence the CD direction from the end station to the AC and thus include it in the LDC.

A further advantage of the EST, CQF, or ATS decoupling of ACs is that the ACs are generally decoupled, not only from an LDC perspective. This is particularly important in the case of plant extensions. In this case, it is often an issue that newly added applications create changed traffic situations in a network. This can have an influence on applications that were previously operated without problems, such that they experience communication problems owing to the additional load. Previously deterministic traffic, could, for example, no longer be deterministic after the extension. This is especially risky with SPQ as a hardware basis. With the EST, CQF, or ATS decoupling possibilities, this problem is avoided as newly added ACs are assigned their own stream classes and thus their own gating windows without influencing the already established ones. Therefore, the bandwidth reserve should be considered when designing the network for an automation plant, that is, in this case, space for possibly new reserve gating windows within the network cycle.

In combination with the network cycle length, one disadvantage or peculiarity of gating-window-based solutions must be discussed at this point. As outlined

in Section 4.7, the fastest application cycle occurrence in the network domain assigns the maximum network cycle length. Shorter network cycles are also possible. However, with the AC decoupling method, all ACs time slots for I-CD and NI-CD as well as the BE phase must fit into this network cycle. This relation is given by:

$$T_{AppMin} \geq \ T_{Netw} \geq T_{GW\ BE} + \sum_{i=1}^{\gamma} (T_{GW\ ICDi} + T_{GW\ NICDi}) \qquad (7\text{-}8)$$

where T_{AppMin} is the smallest application communication cycle in the network domain applying the EST-DLDC, T_{Netw} is the network cycle, $T_{GW\ BE}$ is the gating window length for BE traffic, $T_{GW\ ICDi}$ the gating window length for the I-CD of one AC, $T_{GW\ NICDi}$ the gating window length for the NI-CD of one AC, and γ is the number of ACs in the EST-DLDC network domain. These two contradicting limiting preconditions have conflicting demands regarding the network cycle length. Thus, the selection of the network cycle might not be possible with fast application cycles, many ACs, or large CD traffic demands. A fact that defuses this is that multiple ACs typically appear only in controller-level rings which usually have slower application cycles than a field-level ring.

Nevertheless, should such a conflict be present, a possible solution is to segment the controller level ring as shown in Figure 7.8. However, a necessary precondition for this is that the communication structure of the network allows a sensible separation into clusters of devices, forming separate rings for ACs that communicate more intensively with each other. The inter-communication between these clusters, that is, rings, should be as small as possible which then allows to keep the gating windows of ACs of the other ring small.

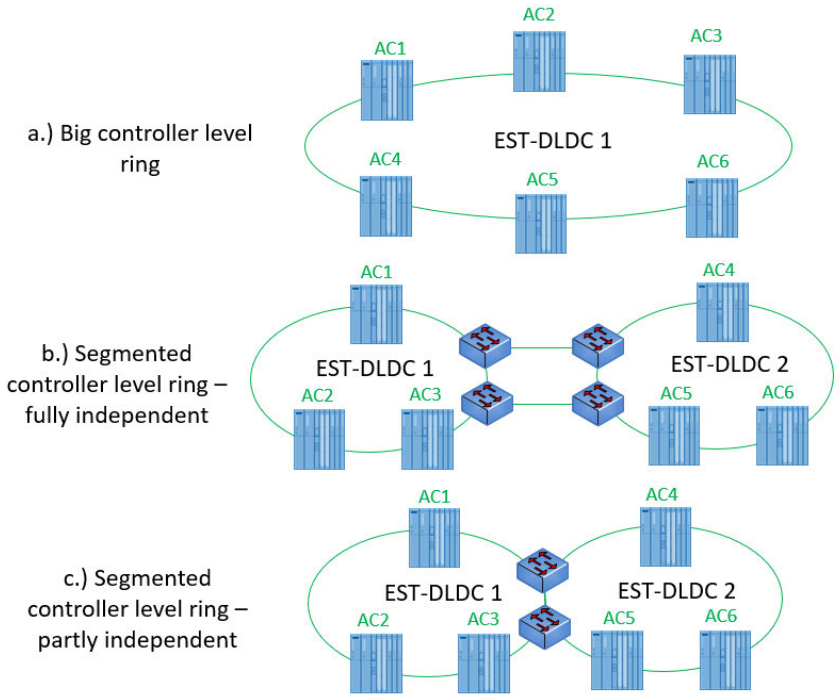


Figure 7.8: Segmentation of the controller level ring

The top-most solution a.) in Figure 7.8 shows a single controller level ring, as it would be set up if the EST-DLDC is possible within the network cycle. Solution b.) shows the segmented controller level ring, resulting in two independent controller level rings that are redundantly interconnected via four bridges. This redundant interconnection can be achieved using suitable redundancy protocols such as RSTP/MSTP or MRP Interconnection to avoid broadcast domain loops on non-stream traffic levels, for example, for BE traffic. CD streams are not forwarded via FDB learning but are under path control and are thus not subject to RSTP/MSTP or MRP Interconnection path-changing redundancy protocol influences. The seamless inter-ring stream traffic uses the seamless FRER protocol for transport. Both rings can use the same communication cycle but with longer gating windows for ACs belong to the ring and smaller windows for those of the other ring. The advantage of solution b.) is that changes or errors in one ring do not influence link loads or load distribution of the other rings. However, its disadvantage is the higher cost for four additional bridges. Solution c.) also shows controller-level ring segmentation, but now with only partly independent rings at the lower cost of

only two interconnection bridges. This can also be achieved by applying either RSTP/MSTP or vendor-specific faster protocols especially tailored for automation purposes and involving shorter reconfiguration timings than RSTP in the case of network errors. However, the disadvantage of solution c.) is that a missing link or device in one ring also induces load changes in the other ring. For example, if the link between AC4 and AC6 is lost, the traffic that used this link must be redirected to the path between AC4 and AC5. However, this path also has a common link with the path between AC1 and AC3, thereby influencing link load distributions of the other ring.

7.5 Performance Considerations for Multiple Automation Controller Solutions

With the **SPQ-DLDC** solution from Subsection 7.4.2, implying mutual load distribution controller dependency, control sovereignty is handed from AC to AC in a round-robin fashion. Each AC calculates its LDC according to the previously described algorithm. It has also been stated that the time to elapse before the token can be handed to the next AC depends on the application cycle of the controlled CD class which is shifted from one path to another because of the optimisation algorithm calculations. This is because the rolling mean measurement interval must have been elapsed for the next controller to measure the usual traffic distribution after the last change. The number of optimisation attempts required for each AC and application cycle class in the network to reach the final optimum load distribution in the network cannot be predicted. This is different for each network and depends on many factors, such as the overall amount of seamless and non-seamless streams, the location of the talkers and listeners, and the network size, that is, the number of nodes. Therefore, CD with long application cycles not only has a lower bandwidth consumption but also lower influence on the load distribution if it is changed. Their changes also require longer token passing time, and thus, a longer overall time to achieve the overall optimisation. Therefore, it is recommended that an SPQ-DLDC controller residing on an AC starts by first checking the CD streams with the fastest application cycle. This ensures a faster convergence time for the overall optimsation result.

With the solutions **EST-DLDC, CQF-DLDC, and ATS-DLDC,** which exclude mutual load distribution controller dependencies, the LDCs residing on the different ACs can work in parallel. This is because each AC works with a dedicated stream class for its CD streams which can then be kept in different transmission slots and gating window slots, as outlined in Subsection 7.4.3. Thus, the distribution load controls also operate in separate time slots and gating windows. Therefore, these three solutions are expected to converge much faster than with SPQ-DLDC method.

Comparing simulations between the two described solutions are pointless for the following reasons:

1. The parallel processing of EST-DLDC, CQF-DLDC, and ATS-DLDC, will in any way be faster than the subsequent processing of the same control loops with SPQ-DLDC.
2. Quantitative convergence speed simulation results would be of low value. This is because they depend on many network parameters such as number of ACs, number of bridges, number of applications, application cycle intervals, and stream sizes of applications.

7.6 Chapter Summary

The optimised load distribution control method for a single AC in the field level-ring from Chapter 6 must be extended to be applied in a controller-level ring for machine-to-machine (M2M) communication with multiple ACs. There are two control strategies possible to control the load of multiple ACs. These are the Central Load Distribution Control (CLDC) and Distributed Load Distribution Control (DLDC). The CLDC is typically located on a CNC, which typically also performs other tasks such as network orchestration, nodes configuration, transmission scheduling, and resource reservations. The advantage of CLDC is that it can achieve an optimum load distribution because it has access to all traffic information and influence on all nodes in the entire network domain. One disadvantage is that constant reorganisation of the network with each change or extension of the setup would constantly disturb the established automation applications. Another disadvantage is the single point of failure of the CNC. Its loss would lead to a loss of the complete load-distribution function. With DLDC, the single load distribution controllers are located on various

ACs and thus distributed in the network. Each AC is responsible only for its own CD traffic. Thus, the failure of one AC has only a limited influence on the overall load distribution optimisation function. For these reasons, DLDC is the preferred solution and is the basis for the solutions proposed in this chapter.

Different traffic shapers and schedulers have different effects on the DLDC approach. With the SPQ as the hardware basis, a mutual dependency exists between the distributed load controllers. A DLDC solution based on SPQ is proposed and outlined, which is based on a step-by-step optimisation process handing over a control sovereignty from AC to AC in a round-robin system. Thus, the ACs SPQ-DLDC is timely decoupled. An algorithm for this SPQ-DLDC was proposed.

A DLDC solution without mutual dependency among the load controllers is possible with EST, CQF and ATS hardware solutions. With all three shapers/schedulers, the time-synchronised controllers and bridges in the network enable a time-synchronised transmission of the CD traffic. In combination with the gating windows of shapers/schedulers, time-multiplexed forwarding within the network is possible. Each AC uses different stream classes for its CD, which are assigned to different time slots; thus, the ACs with their LDCs are decoupled and run in parallel. The EST-DLDC is preferable to the CQF-DLDC and ATS-DLDC for its lower end-to-end latency, allowing less complex flow control setups. The SPQ-DLDC makes better use of the available bandwidth as it lacks fixed reserved time slots and allows flexible bandwidth usage by all the ACs. In contrast, all ACs LDCs can operate in parallel when using EST-DLDC, CQF-DLDC, or ATS-DLDC. Thus, these have a convergence speed advantage compared to the SPQ-DLDC.

A comparative simulation of the two solutions is deliberately omitted because, by comparing parallel processing in contrast to sequential processing, the parallel solution with an identical network setup is expected to be faster. The speed differences will be different for each network detail difference. Thus, a simulation would have a low orientation value. Therefore, it is not considered worthwhile.

Chapter 8 Conclusion and Further Work

This chapter concludes the thesis with a summary of what has been achieved regarding the research objectives, as stated in the introduction. It also provides a presentation of the original contribution to the body of knowledge and an outlook for future work.

8.1 Conclusion

This thesis investigates the possibilities of load distribution control (LDC) in the complex TSN MAN landscape. It consists of four main chapters to answer the four research objectives. Each provides an original research work to contribute to the goal of finding solutions for an optimum distributed dynamic LDC method for a TSN MAN.

In the first main chapter, Chapter 4, TSN MANs are analysed in terms of their potential to facilitate LDC systems. It shows that the prevalent TSN MAN topologies are ring and redundantly coupled rings. The best path establishment methods for the physical topology VLANs are SPBV for administration-involved assignment and ISIS-PCR for automated assignment. A distributed LDC approach is better suited for real dynamic load control. This should be located in an influential AC rather than in a bridge. Data priority or traffic class assignment strongly depend on the use of traffic shapers and schedulers. However, these are typically predetermined by the selection of the automation technology. Only a non-seamlessly transported CD is available for LDC, as the reduction in throughput is not an alternative for CD. This is also the reason that only distribution control and flow control are considered relevant for this study and no fairness flow reduction. Linear dynamic control is the most promising control method for TSN MAN because of the typically constant and known ingress data rates, fast communication cycles, and the small dead times of EST and SPQ which are preferred here. It should be combined with traffic engineering during the network planning and setup phase. The load distribution control task is based on the optimisation goal to minimise the maximum load peak along two possible paths within the automation ring. The flow control loop is a sub control loop of the complete distribution control. For the design of this, the application cycle time plays an

important role because the slowest application in the network domain assigns the minimum integration interval for data flow measurement. For stream resource reservation, it is recommended to work with pre-reserved reservations to fulfill demanding dynamic requirements. Furthermore, it reveals that network errors are best handled using pre-reservation or dynamic re-reservation. Thus, Chapter 4 answers the main part of the first research objective to clarify the applicability of load distribution in TSN MANs. It also provides the foundations for answering the subsequent research objectives by clarifying the boundary conditions for LDC within MANs.

The second research objective is addressed in Chapter 5 . The different influences of the traffic shapers and schedulers on data flow control were analysed and simulated. It is shown that EST, SPQ, and SPQ with Preemption are the best selections from a data flow control point of view for bigger TSN MAN or such comprising fast automation applications. CQF and ATS are only recommended for smaller TSN MAN with slower automation applications.

In Chapter 6, the third research objective is developed. It proposes a new dedicated control method, that is optimised for TSN MANs with different application cycles of the automation tasks. Providing dedicated flow controllers for data flows of different application cycle times guarantees an optimised settling time for each application cycle time. This approach also implies a dedicated, per-application cycle class, rolling mean measurement of the CD traffic on each bridge or bridged end station. This design was verified by a network simulation. It is shown that multiple application-cycle-class-dedicated control achieves a faster overall load distribution convergence than a single common flow controller.

In the fourth and last main chapter, Chapter 7, the single controller solution is expanded to multiple controller solutions. Thereby, this chapter provides answers to the fourth research question. Two types of distributed load distribution control solutions for multiple ACs are possible. The essential differentiating criterion is the existence or lack of a mutual controller dependency. Concrete implementations were proposed for both types. The Strict Priority Queuing based distributed load distribution control (SPQ-DLDC) is the only representative of the type that has an

inherent mutual controller dependency. Mutual dependency must be seen as a disadvantage as it complicates control. However, a main advantage of SPQ-DLDC is its flexible bandwidth use, as it does not depend on gating windows, nor does it require a synchronised network cycle or synchronised end stations. With EST, CQF, and ATS, it is possible to build EST-DLDC, CQF-DLDC, and ATS-DLDC without mutual controller dependencies. This reduces the effort required for the multiple controller solution to the same effort that is necessary for a single load distribution controller without any dedicated algorithms to cope with mutual dependency. Comparing EST-DLDC, CQF-DLDC, and ATS-DLDC, EST-DLDC is clearly preferred because of its very low end-to-end latency and, thus, associated more dynamic and economic control design possibilities. The crucial property for these DLDCs is that the bridges and end stations must be time-synchronised. Combined with an AC-dedicated bandwidth-use measurement, a complete separation of the single ACs LDC can be achieved.

8.2 The Contribution to the New Knowledge Generation

The main contribution of this thesis is the presentation of possible solutions for an optimum load distribution control in manufacturing automation networks. In particular:

- An analysis of the broad field of TSN automation networks with its technological diversity such as different TSN traffic shapers and schedulers, control location design possibilities, relevant network topologies, control setup design possibilities and characteristics, eligible traffic classes, automation applications influence, the role of stream reservation, and error mitigation strategies regarding their suitability and influence for load distribution control purposes. Thereby, a comprehensive picture of LDC possibilities within TSN MAN is provided.
- A closed-loop load distribution control model for automation ring networks has been established and the influence of the different types of TSN traffic shapers and schedulers on data flow control are shown. Furthermore, a strong influence of the applications is demonstrated. Their application cycles determine the dynamics and stability of the load distribution control. The findings are validated through simulations.

- Recommendations are given for the use of TSN traffic shapers and schedulers in various TSN MAN. SPQ, SPQ with Preemption, and EST are the best shaper and scheduler selections in high dynamic networks and also in larger networks. CQF and ATS are to be preferred for larger networks with a high hop count or networks containing slow applications.
- An optimised control method for load distribution control in automation networks is proposed. It considers the influence of the applications by providing dedicated load distribution controllers per application cycles or groups of application cycles. This achieves more dynamic load distribution convergence and robustness towards an appearance of new and slow applications. An ns-3 simulation code is provided, and the assumptions are validated through simulations.
- Two load distribution control methods for multiple load distribution controllers residing on multiple ACs are provided. The first method, SPQ-DLDC. represents a solution based on SPQ traffic shaping, with its inherent mutual controller dependencies. The proposed method with its consecutive control sovereignty approach resolves these dependencies. The second method is suitable for EST, CQF and ATS schedulers and shapers which demand time-synchronised end stations, bridges, and bridged end stations. It achieves fast parallel controller calculations without mutual dependencies.

8.3 Limitations and Further Work

The prevailing automation solutions today are mostly application specific physical automation islands. This will continuously shift towards virtual applications running in virtual automation controllers hosted by industrial data centers or edge appliances. This requires increasing deterministic data communication from higher-level automation clouds or data centers down to edge controllers and field devices. Regarding load distribution, this also implies load distribution concepts not only on a horizontal basis within field-level communication rings or controller-level communication rings but also on a vertical basis between and through the hierarchical automation communication layers.

To take this development into account and to provide a fully applicable load distribution control for industrial manufacturing automation, further work that exceeds the focus and possibilities of this thesis is necessary. This would be in particular:

- A redundant ring interconnection exists between a controller-level ring and a field-level ring. If the controller level ring is segmented for the application of EST-DLDC as described in Subsection 7.4.3, then also a ring interconnection topology is part of the network topology. Each ring forms then a separate EST-DLDC domain. However, inter-ring communication will also be involved. The LDC concept should be expanded to cover such ring interconnection topologies.
- For the control design of the application cycle class specific load distribution controllers, it is assumed that the AC has a CD traffic load within each application cycle traffic class to distribute. This may not always be the case in practice. The packet handler controls the number of data packets of data streams to be distributed. Therefore, a more sophisticated packet controller, capable of making distribution decisions across application cycle classes is necessary in the future.
- For the EST-DLDC, a dynamic adaptation of the gating window lengths during runtime, depending on the traffic measurement statistics results, for better bandwidth use would be advantageous.
- In Subsection 7.4.2, an optimsation algorithm for the SPQ-DLDC is proposed. This could stop at the first found improvement and bring it to the network. The alternative would be to search for the best improvement by calculating all possibilities. Comparative simulations on which strategy is advantageous, depending on various networks and traffic setups, would be useful.

Chapter 9 Bibliography

Ahmad, I., Karunarathna, S. N., Ylianttila, M., & Gurtov, A. (2015). Load Balancing in Software Defined Mobile Networks. In *Software Defined Mobile Networks (SDMN) : Beyond LTE Network Architecture* (pp. 225-245). John Wiley & Sons, Ltd : Chichester, UK. https://doi.org/10.1002/9781118900253.ch13

Ahmad, M., & Khan, R. Z. (2018). Load Balancing Tools and Techniques in Cloud Computing: A Systematic Review. In (pp. 181-195). https://doi.org/10.1007/978-981-10-3773-3_18

Ali, M., Chiruvolu, G., & Ge, A. (2005). Traffic engineering in metro Ethernet. *IEEE Network*, *19*(2). https://doi.org/10.1109/MNET.2005.1407693

Alvarez Vadillo, I., Ballesteros, A., Barranco, M., Gessner, D., Djerasevic, S., & Proenza, J. (2019). Fault Tolerance in Highly Reliable Ethernet-Based Industrial Systems. *Proceedings of the IEEE*, *107*(6), 977-1010. https://doi.org/10.1109/JPROC.2019.2914589

Anna Victoria Oikawa, C. R., Freitas, V., Castro, M., & Pilla, L. L. (2020). Adaptive Load Balancing based on Machine Learning for Iterative Parallel Applications. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (pp. 94-101). IEEE. https://doi.org/10.1109/PDP50117.2020.00021

ANSI/TIA-568.1-D. (2015). Commercial Building Telecommunications Cabling Standard. In. Washington, D.C., USA: American National Standards Institute.

Antic, M., Maksic, N., Knezevic, P., & Smiljanic, A. (2010). Two phase load balanced routing using OSPF. *IEEE Journal on Selected Areas in Communications*, *28*(1). https://doi.org/10.1109/JSAC.2010.100106

Åström, K. J. (2012). *Introduction to Stochastic Control Theory*. Dover Publications.

Avnu. (2023). *Avnu Alliance*. Retrieved 2023/08/10 from https://avnu.org

Bensaou, B., Tsang, D. H. K., & King Tung, C. (2001). Credit-based fair queueing (CBFQ): a simple service-scheduling algorithm for packet-switched networks. *IEEE/ACM Transactions on Networking*, *9*(5). https://doi.org/10.1109/90.958328

Beran, J., Fiedler, P., & Zezulka, F. (2010). Virtual Automation Networks. *IEEE Industrial Electronics Magazine*, *4*(3). https://doi.org/10.1109/MIE.2010.937930

Bojović, P. D., & Živko, B. (2022). Hybrid SDN Networks: A Multi-parameter Server Load Balancing Scheme. *Journal of Network and Systems Management*, *30*(2). https://doi.org/10.1007/s10922-022-09642-y

Bolla, R., Davoli, F., Maryni, P., & Parisini, T. (1998). An adaptive neural network admission controller for dynamic bandwidth allocation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *28*(4). https://doi.org/10.1109/3477.704298

Bonomi, F., & Fendick, K. W. (2002). The rate-based flow control framework for the available bit rate ATM service. *IEEE Network*, *9*(2). https://doi.org/10.1109/65.372653

Braun, R. (1997). Internet protocols for multimedia communications. II. Resource reservation, transport, and application protocols. *IEEE MultiMedia*, *4*(4). https://doi.org/10.1109/93.641882

Cardellini, V., Colajanni, M., & Yu, P. S. (1999). Dynamic load balancing on Web-server systems. *IEEE Internet Computing*, *3*(3). https://doi.org/10.1109/4236.769420

Chadha, A., & Gupta, A. K. (2013). Attaining more Efficiency from Enhanced Interior Gateway Routing Protocol. *International Journal of Computer Applications*, *975*, 8887.

Chaturvedi, D. K. (2017). *Modeling and simulation of systems using MATLAB and Simulink*. CRC press.

Chen, R. R., & Khorasani, K. (2011). A robust adaptive congestion control strategy for large scale networks with differentiated services traffic. *Automatica*, *47*(1), 26-38. https://doi.org/10.1016/j.automatica.2010.08.019

Chiu, D.-M., & Jain, R. (1989). Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks & ISDN Systems*, *17*(1), 1.

Cisco. (2022). *How Does Load Balancing Work?* Cisco Systems. Retrieved 2023/09/20 from http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5212-46.html

Comet. (2023). *COMET Optimisation Technology*. Comet. Retrieved 2023/09/18 from https://www.comet.com/docs/v2/

Craciunas, S. S., Serna, R., Martin, O., & Steiner, C. W. (2016). Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks. *ACM International Conference Proceeding Series*, *19-21-October-2016*, 183-192. https://doi.org/10.1145/2997465.2997470

Crotty, M. (1998). *The foundations of social research : meaning and perspective in the research process*. Sage Publications.

De Cicco, L., Mascolo, S., & Niculescu, S.-I. (2011). Robust stability analysis of Smith predictor-based congestion control algorithms for computer networks. *Automatica*, *47*(8), 1685-1692. https://doi.org/10.1016/j.automatica.2011.02.036

de Sousa, A., & Soares, G. (2007). Improving load balance and minimizing service disruption on ethernet networks with IEEE 802.1 S MSTP. Workshop on IP QoS and traffic control,

de Sousa, A., & Soares, G. (2008). *Improving Load Balance and Minimizing Service Disruption on Ethernet Networks with IEEE 802.1 S MSTP*. EuroFGI Workshop on IP QoS and Traffic Control.

Duriez, T., Brunton, S. L., & Noack, B. (2017). *Machine Learning Control - Taming Nonlinear Dynamics and Turbulence* (1st ed. 2017 ed.). Springer International Publishing.

Elwalid, A., Jin, C., Low, S., & Widjaja, I. (2002). MATE: multipath adaptive traffic engineering. *Computer Networks*, *40*(6), 695-709. https://doi.org/10.1016/S1389-1286(02)00308-0

Falk, J., Hellmanns, D., Carabelli, B., Nayak, N., Durr, F., Kehrer, S., . . . International Conference on Networked Systems Munich, G. M. M. (2019). NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++. In *2019 International Conference on Networked Systems (NetSys)* (pp. 1-8). https://doi.org/10.1109/NetSys.2019.8854500

Farahmand, F., Qiong, Z., & Jue, J. P. (2005). A closed-loop rate-based contention control for optical burst switched networks. In *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.* (pp. 5-1993). IEEE. https://doi.org/10.1109/GLOCOM.2005.1578014

Farkas, J., Bello, L. L., & Gunther, C. (2018). Time-Sensitive Networking Standards. *IEEE Communications Standards Magazine*, *2*(2). https://doi.org/10.1109/MCOMSTD.2018.8412457

Finn, N. (2018). Introduction to Time-Sensitive Networking. *IEEE Communications Standards Magazine*, *2*(2). https://doi.org/10.1109/MCOMSTD.2018.1700076

Finn, N. (2019). Multiple Cyclic Queuing and Forwarding. In.

Fortz, B., Rexford, J., & Thorup, M. (2002). Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, *40*(10). https://doi.org/10.1109/MCOM.2002.1039866

Fortz, B., & Thorup, M. (2000). Internet traffic engineering by optimizing OSPF weights. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications.*

Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (pp. 519-528). https://doi.org/10.1109/INFCOM.2000.832225

Gavrilut, V., Zhao, L., Raagaard, M. L., & Pop, P. (2018). AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN. *IEEE Access*, *PP*(99). https://doi.org/10.1109/ACCESS.2018.2883644

Gebali, F. (2015). *Analysis of computer and communication networks* (Second edition. ed.). Springer. https://doi.org/10.1007/978-3-319-15657-6

Geist, M., & Jaeger, B. (2019). Overview of TCP Congestion Control Algorithms.

Gillani, B., Kent, R., & Aggarwal, A. (2005). Topology Reconfiguration Mechanism for Traffic Engineering in WDM Optical Network. In (pp. 161-169). IEEE,; 2005).

Goodwin, G. C., Graebe, S. F., & Salgado, M. E. (2001). *Control system design* (Vol. 240). Prentice Hall Upper Saddle River.

Goulamghoss, M. I., & Bassoo, V. (2020). Analysis of traffic engineering and fast reroute on multiprotocol label switching. *Journal of Ambient Intelligence and Humanized Computing*, *12*(2), 2409-2420. https://doi.org/10.1007/s12652-020-02365-5

Grigorjew, A., Baier, C., Metzger, F., & Hosfeld, T. (2021). Distributed Implementation of Deterministic Networking in Existing Non-TSN Ethernet Switches. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)* (pp. 1-6). IEEE. https://doi.org/10.1109/ICCWorkshops50388.2021.9473776

Grix, J. (2019). *The foundations of research* (Third edition. ed.). Red Globe Press.

Grosu, D., & Chronopoulos, A. T. (2005). Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, *65*(9), 1022-1034. https://doi.org/10.1016/j.jpdc.2005.05.001

Hagan, M. T., Demuth, H. B., & Jesús, O. D. (2002). An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control*, *12*(11), 959-985. https://doi.org/10.1002/rnc.727

Han, Z., Kong, J., Wang, Z., Zhang, Y., Liu, K., Pan, L., . . . Wu, D. (2021). AI-based network topology optimization system. *ITU Journal on Future and Evolving Technologies*, *2*(4), 81-90. https://doi.org/10.52953/YXTB5085

Hasegawa, G., Murata, M., & Miyahara, H. (2000). Fairness and stability of congestion control mechanisms of TCP. *Telecommunication Systems*, *15*(1-2), 167-184.

Hellmanns, D., Falk, J., Glavackij, A., Hummen, R., Kehrer, S., Durr, F., & Ieee International Conference on Industrial Technology Buenos Aires, A. F. F. (2020). On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN. In *2020 IEEE International Conference on Industrial Technology (ICIT)* (pp. 298-303). IEEE. https://doi.org/10.1109/ICIT45562.2020.9067122

Henderson, T., & Imputato, P. (2023). Proceedings of the 2023 Workshop on ns-3. Arlington, VA, USA.

Ho, T. V., Deville, Y., Bonaventure, O., & François, P. (2011). Traffic engineering for multiple spanning tree protocol in large data centers. In. 23rd International Teletraffic, Congress.

Huang, Y., Wang, S., Zhang, X., Huang, T., & Liu, Y. (2022). Flexible Cyclic Queuing and Forwarding for Time-Sensitive Software-Defined Networks. *IEEE Transactions on Network and Service Management*, *PP*(99). https://doi.org/10.1109/TNSM.2022.3198171

Huawei. (2010). *Shortest Path Bridging IEEE 802.1aq Tutorial and Demo*. Retrieved 2023/09/10 from https://archive.nanog.org/meetings/nanog50/presentations/Sunday/IEEE_8021aqShortest_Path.pdf

IEC 61158-5-10. (2023). Industrial communication networks –Fieldbus specifications. In *Part 5-10: Application layer service definition – Type 10 elements*: International Electrotechnical Commission.

IEC 61158-6-10. (2023). Industrial communication networks –Fieldbus specifications. In *Part 6-10: Application layer protocol specification – Type 10 elements*: International Electrotechnical Commission.

IEC 62439-2. (2021). Industrial communication networks - High availability automation networks - Part 2: Media Redundancy Protocol (MRP). In: International Electrotechnical Commission.

IEC 62439-3. (2021). Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR) In: International Electrotechnical Commission.

IEC/IEEE 60802. (2018). TSN Profile for Industrial Automation, Use Cases. In.

IEEE 802.1AS. (2020). IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications. In. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1BA. (2011). IEEE Standard for Local and Metropolitan Area Networks—Audio Video Bridging (AVB) Systems. In. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1CB. (2017). IEEE Standard for Local and Metropolitan Area Networks—Frame Replication and Elimination for Reliability. In. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1CS. (2019). Link-local Registration Protocol (LRP). In. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Q. (2022). IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks. In. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Q TSN TG. (2022). Time-Sensitive Networking (TSN) Task Group. In: Institute of Electrical and Electronic Engineers.

IEEE 802.1Qav. (2009). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. In *Amendment: Forwarding and Queuing Enhancements for Time-Sensitive Streams*. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qbu. (2015). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. In *Amendment: Frame Preemption*. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qbv. (2015). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. In *Amendment: Enhancements for Scheduled Traffic*. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qca. (2015). IEEE Standard for Local and Metropolitan Area Networks—Bridges and Virtual Bridged Local Area Networks. In *Amendment: Path Control and Reservation* New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qcc. (2018). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. In *Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements* New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qch. (2019). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. In *Amendment 29: Cyclic Queuing and Forwarding* New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qci. (2016). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks. In *Amendment: Per-Stream Filtering and Policing* New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qcr. (2020). IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks. In *Amendment: Asynchronous Traffic Shaping* New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.1Qdd. (2023). IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks. In *Amendment: Resource Allocation Protocol* New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 802.3br. (2016). IEEE Standard for Ethernet In *Amendment 5: Specification and Management Parameters for Interspersing Express Traffic*. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IEEE 1588. (2019). IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In. New York, USA: Institute of Electrical and Electronics Engineers (IEEE).

IETF RFC 793. (1981). Ttansmission Control Protocol. In *Request for Comments: 793*. USA, California: Information Sciences Institute, University of Southern California.

IETF RFC 2001. (1997). TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. In *Request for Comments: 2001*: Internet Engineering Task Force.

IETF RFC 2205. (1997). Resource ReSerVation Protocol (RSVP). In *Request for Comments: 2205*: Internet Engineering Task Force.

IETF RFC 3630. (2003). Traffic Engineering (TE) Extensions to OSPF Version 2. In *Request for Comments: 3630*: Internet Engineering Task Force.

IETF RFC 5305. (2008). IS-IS Extensions for Traffic Engineering. In *Request for Comments: 5305*: Internet Engineering Task Force.

IETF RFC 5681. (2009). TCP Congestion Control. In *Request for Comments: 5681*. USA, Indiana: Purdue University.

IETF RFC 6002. (2010). Generalized MPLS (GMPLS) Data Channel Switching Capable (DCSC) and Channel Set Label Extensions. In *Request for Comments: 6002*: Internet Engineering Task Force.

IETF RFC 6582. (2012). The NewReno Modification to TCP's Fast Recovery Algorithm. In *Request for Comments: 6582*. Finland, Oulu: Oulu University.

IETF RFC 8570. (2019). IS-IS Traffic Engineering (TE) Metric Extensions. In *Request for Comments: 8570*: Internet Engineering Task Force.

IETF RFC 8578. (2019). Deterministic Networking Use Cases. In *Request for Comments: 8578*: Internet Engineering Task Force.

IETF RFC 8655. (2019). Deterministic Networking Architeture. In *Request for Comments: 8655*: Internet Engineering Task Force.

ISO/IEC 7498-1. (2000). Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. In: International Organization for Standardization/ International Electrotechnical Commission.

Jahde, S., Sakhtivel, S., Sudha, R. V., Verma, R. K., Walia, R., & Lokesh, M. R. (2021). SDN Network Load Balancing using Environmental Congenital ACO Methodology. *International Journal of Biology, Pharmacy and Allied Sciences (IJBPAS)*, *10*. https://doi.org/10.31032/ijbpas/2021/10.11.1079

Jain, R. (1998). Congestion control and traffic management in ATM networks: Recent advances and a survey. *Computer Networks & ISDN Systems*, *28*.

Jong-Moon, C. (2000). Analysis of MPLS traffic engineering. In *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems (Cat.No.CH37144)* (pp. 550-553). https://doi.org/10.1109/MWSCAS.2000.952816

Kanagarathinam, M. R., Singh, S., Sandeep, I., Kim, H., Maheshwari, M. K., Hwang, J., . . . Saxena, N. (2020). NexGen D-TCP: Next Generation Dynamic TCP Congestion Control Algorithm. *IEEE Access*, *8*. https://doi.org/10.1109/ACCESS.2020.3022284

Kandula, S., Katabi, D., S. Davie, B., & Charny, A. (2005). *Walking the tightrope: Responsive yet stable traffic engineering* (Vol. 35). Computer Communication Review. https://doi.org/10.1145/1080091.1080122

Kandula, S., Katabi, D., Sinha, S., & Berger, A. (2007). Dynamic Load Balancing Without Packet Reordering. *COMPUTER COMMUNICATION REVIEW*, *37*(2), 51-62.

Kasoro, N. M., Kasereka, S. K., Alpha, G. K., & Kyamakya, K. (2021). ABCSS: A novel approach for increasing the TCP congestion window in a network. *Procedia Computer Science*, *191*, 437-444. https://doi.org/10.1016/j.procs.2021.07.075

Kaszkurewicz, E. (2010). A proposed solution for the load balancing problem on heterogeneous clusters based on a delayed neural network. *International Journal of Intelligent Computing and Cybernetics*, *3*(1), 73-93. https://doi.org/10.1108/17563781011028550

Katabi, D., Handley, M., & Rohrs, C. (2002). Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM Computer Communication Review*, *32*(4), 89. https://doi.org/10.1145/964725.633035

Katyal, M., & Mishra, A. (2014). A Comparative Study of Load Balancing Algorithms in Cloud Computing Environment. *International Journal of Distributed and Cloud Computing*, *1*(2).

Kelly, F. P., Maulloo, A., & Tan, D. (1998). Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, *49*, 237-252.

Kirrmann, H., & Dzung, D. (2006). Selecting a Standard Redundancy Method for Highly Available Industrial Networks. In *2006 IEEE International Workshop on Factory Communication Systems* (pp. 386-390). IEEE. https://doi.org/10.1109/WFCS.2006.1704184

Lemeshko, O., Vavenko, T., & Ovchinnikov, K. (2013). Design of multipath routing scheme with load balancing in MPLS-network. In *12th International Conference on the Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)* (pp. 211-213). IEEE.

Li, Y., Jiang, J., & Hong, S. H. (2022). Joint Traffic Routing and Scheduling Algorithm Eliminating the Nondeterministic Interruption for TSN Networks Used in IIoT. *IEEE Internet of Things Journal*, *9*(19). https://doi.org/10.1109/JIOT.2022.3163411

Linux. (2023). *Load Balancing*. LinuxVirtualServer.org. Retrieved 2023/09/08 from http://kb.linuxvirtualserver.org/wiki/Load_balancing#Computing_Load_Balancing

LNI4.0. (2023). *Labs Network Industrie 4.0*. Retrieved 2023/08/10 from https://lni40.de/

Lo Bello, L., & Steiner, W. (2019). A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. *Proceedings of the IEEE*, *107*(6), 1094-1120. https://doi.org/10.1109/JPROC.2019.2905334

Lopez-Perez, D., Laselva, D., Wallmeier, E., Purovesi, P., Lunden, P., Virtej, E., . . . Ding, M. (2016). Long Term Evolution-Wireless Local Area Network Aggregation Flow Control. *IEEE Access*, *4*. https://doi.org/10.1109/ACCESS.2016.2643690

Lopez, V., Hernandez, J. A., Gonzalez de Dios, O., Fernandez Palacios, J., & Aracil, J. (2010). Multilayer Traffic Engineering for IP Over WDM Networks Based on Bayesian Decision Theory. *IEEE/OSA Journal of Optical Communications and Networking*, *2*(8). https://doi.org/10.1364/JOCN.2.000515

Lu, Y., Fu, B., Xi, X., Zhang, Z., & Zhang, N. (2018). Medium Rate Control Method for Ship Mobile Network Traffic Generation. *Journal of Coastal Research*, *83*(sp1), 261-266. https://doi.org/10.2112/SI83-042.1

Ma, Y.-W., Chen, J.-L., Tsai, Y.-H., Cheng, K.-H., & Hung, W.-C. (2017). Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking. *Wireless Personal Communications : An International Journal*, *94*(4), 3549-3574. https://doi.org/10.1007/s11277-016-3790-y

Mandić, Z., Stankovski, S., Ostojić, G., & Popović, B. (2022, 16-18 March 2022). Potential of Edge Computing PLCs in Industrial Automation. 2022 21st International Symposium INFOTEH-JAHORINA (INFOTEH),

Mascolo, S. (2000). Smith's principle for congestion control in high-speed data networks. *IEEE Transactions on Automatic Control*, *45*(2). https://doi.org/10.1109/9.839966

Matía, F., Marichal, G. N. s., & Jiménez, E. (2014). *Fuzzy modeling and control : theory and applications*. Atlantis Press. https://doi.org/10.2991/978-94-6239-082-9

Metawei, M. A., Ghoneim, S. A., Haggag, S. M., & Nassar, S. M. (2012). Load balancing in distributed multi-agent computing systems. *Ain Shams Engineering Journal*, *3*(3), 237-249. https://doi.org/10.1016/j.asej.2012.03.001

Microsoft. (2023). *Network Load Balancing*. Microsoft. Retrieved 2023/09/22 from https://docs.microsoft.com/en-us/windows-server/networking/technologies/network-load-balancing

Mohammadnia, A., Rahmani, R., Mohammadnia, S., & Bekravi, M. (2016). A Load Balancing Routing Mechanism Based on Ant Colony Optimization Algorithm for Vehicular Adhoc Network. In (ISSN 0975-6485 Volume 7, Number 1 (2016) ed.). (International Journal Network and Computer Engineering.)

Mowei, W., Yong, C., Xin, W., Shihan, X., & Junchen, J. (2018). Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Network*, *32*(2). https://doi.org/10.1109/MNET.2017.1700200

Müller, A. C., & Guido, S. (2017). *Introduction to machine learning with Python : a guide for data scientists* (First edition. ed.). O'Reilly Media, Inc.

Nam-Uk, K., Hyun-Su, L., Hong-Shik, P., & Minho, K. (2009). Traffic Load Distribution-Based Excess Bandwidth Allocation in Time-Division-Multiplexed PONs. *Journal of Lightwave Technology*, *27*(19). https://doi.org/10.1109/JLT.2009.2022768

Nasrallah, A. (2019). Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS). *IEEE Access*, *7*, 44165-44181.

Nasrallah, A., Thyagaturu, A., Alharbi, Z., Wang, C., Shao, X., Reisslein, M., & El Bakoury, H. (2019). Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys & Tutorials*, *PP*(99). https://doi.org/10.1109/COMST.2018.2869350

Nayak, N. (2018). Scheduling & routing time-triggered traffic in time-sensitive networks. *Dissertation*(4), 176.

Neely, M. J., Li, C. P., & Modiano, E. (2008). Fairness and optimal stochastic control for heterogeneous networks. *IEEE/ACM Transactions on Networking*, *16*(2), 396-409. https://doi.org/10.1109/TNET.2007.900405

Nezami, Z., Zamanifar, K., Djemame, K., & Pournaras, E. (2021). Decentralized Edge-to-Cloud Load Balancing: Service Placement for the Internet of Things. *IEEE Access*, *9*. https://doi.org/10.1109/ACCESS.2021.3074962

Nikolic, B., Yomsi, P. M., & Ojewale, M. A. (2020). Multi-Level Preemption in TSN: Feasibility and Requirements Analysis. In *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)* (pp. 47-55). https://doi.org/10.1109/ISORC49007.2020.00017

Normey-Rico, J. E., & Camacho, E. F. (2007). *Control of Dead-time Processes* (Online-ausg. ed.). Springer-Verlag London Limited. https://doi.org/10.1007/978-1-84628-829-6

ns-3. (2023). *The ns-3 Network Simulator*. Retrieved 2021/09/10 from https://www.nsnam.org/

ODVA. (2023). EtherNet/IP - Technology Overview Series. In.

Ohnishi, H., Okada, T., & Noguchi, K. (1988). Flow control schemes and delay/loss tradeoff in ATM networks. *IEEE Journal on Selected Areas in Communications*, *6*(9), 1609-1616.

Ojewale, M. A., & Yomsi, P. M. (2020). Routing heuristics for load-balanced transmission in TSN-based networks. *ACM SIGBED Review*, *16*(4), 20-25. https://doi.org/10.1145/3378408.3378411

Otoshi, T., Ohsita, Y., Murata, M., Takahashi, Y., Ishibashi, K., & Shiomoto, K. (2015). Traffic prediction for dynamic traffic engineering. *Computer Networks*, *85*, 36. https://doi.org/10.1016/j.comnet.2015.05.001

Patan, K. (2015). Neural Network-Based Model Predictive Control: Fault Tolerance and Stability. *IEEE Transactions on Control Systems Technology*, *23*(3). https://doi.org/10.1109/TCST.2014.2354981

Patino, H. D., & Liu, D. (2000). Neural network-based model reference adaptive control system. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *30*(1). https://doi.org/10.1109/3477.826961

Perry, J., Ousterhout, A., Balakrishnan, H., Shah, D., & Fugal, H. (2014). Fastpass a centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM conference on SIGCOMM* (pp. 307-318). https://doi.org/10.1145/2619239.2626309

Pompili, D., & Priscoli, F. D. (2008). A closed-loop fuzzy traffic controller for fair bandwidth sharing. *ACM SIGBED Review*, *5*(2), 1-6. https://doi.org/10.1145/1399583.1399588

Prabakaran, S., & Ramar, R. (2021). Software Defined Network: Load Balancing Algorithm Design and Analysis. *The International Arab Journal of Information Technology*, *18*(3). https://doi.org/10.34028/iajit/18/3/7

Prabhavat, S., Nishiyama, H., Ansari, N., & Kato, N. (2012). On Load Distribution over Multipath Networks. *IEEE Communications Surveys & Tutorials*, *14*(3). https://doi.org/10.1109/SURV.2011.082511.00013

Puqi Perry, T., & Tai, T. Y. C. (1999). Network traffic characterization using token bucket model. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)* (pp. 51-62). https://doi.org/10.1109/INFCOM.1999.749252

Puype, B., Colle, D., Pickavet, M., & Demeester, P. (2009). Multilayer traffic engineering for multiservice environments. *Photonic Network Communications*, *18*(2), 150-159. https://doi.org/10.1007/s11107-008-0179-1

Quang, P. T. A., Magnouche, Y., Leguay, J., Gong, X., & Zeng, F. (2020). Model predictive control for load balancing. In *Proceedings of the SIGCOMM '20 Poster and Demo Sessions* (pp. 37-38). https://doi.org/10.1145/3405837.3411383

Räcke, H. (2009, 2009//). Survey on Oblivious Routing Strategies. Mathematical Theory and Computational Practice, Berlin, Heidelberg.

Rajeshkannan, R., & Aramudhan, M. (2016). Comparative study of load balancing algorithms in cloud computing environment. *Indian Journal of Science and Technology*, *9*(20). https://doi.org/10.17485/ijst/2016/v9i20/85866

Ranjan, R., Villari, M., Fazia, M., Jayaraman, P. P., & Georgakopoulos, D. (2016). Internet of Things and Edge Cloud Computing Roadmap for Manufacturing. *IEEE Cloud Computing*, *3*(4), 66-73. https://doi.org/10.1109/MCC.2016.91

Russell, S. J., & Norvig, P. (2021). *Artificial intelligence : a modern approach* (Fourth edition. ed.). Pearson Education.

Saedi, T., & El-Ocla, H. (2021). TCP CERL+: revisiting TCP congestion control in wireless networks with random loss. *Wireless Networks*, *27*(1), 423-440. https://doi.org/10.1007/s11276-020-02459-0

Santos, D., de Sousa, A., Alvelos, F., Dzida, M., Pioro, M., & Zagozdzon, M. (2009). Traffic Engineering of Multiple Spanning Tree Routing Networks: the Load Balancing Case. In *2009 Next Generation Internet Networks* (pp. 1-8). https://doi.org/10.1109/NGI.2009.5175784

Shahid, M. A., Islam, N., Alam, M. M., Su'ud, M. M., & Musa, S. (2020). A Comprehensive Study of Load Balancing Approaches in the Cloud Computing Environment and a Novel Fault Tolerance Approach. *IEEE Access*, *8*. https://doi.org/10.1109/ACCESS.2020.3009184

Shuo, W., Jiao, Z., Tao, H., Tian, P., Jiang, L., & Yunjie, L. (2016). FDALB: Flow distribution aware load balancing for datacenter networks. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)* (pp. 1-2). IEEE. https://doi.org/10.1109/IWQoS.2016.7590409

Singh, A., Kumar, A., & Chauhan, B. K. (2022). A Comprehensive Study of Edge Computing and the Impact of Distributed Computing on Industrial Automation. Cognitive Informatics and Soft Computing, Singapore.

Smith, S. (2003). *Introduction to MPLS*. Cisco Systems. Retrieved 2023/09/22 from https://www.cisco.com/c/dam/global/fr_ca/training-events/pdfs/Intro_to_mpls.pdf

Soldatos, J., Lazaro, O., & Cavadini, F. (2019). *The digital shopfloor : industrial automation in the industry 4.0 era : performance analysis and applications*. River Publishers. https://doi.org/10.1201/9781003339717

Song, Y., Guo, C., Xu, P., Li, L., & Zhang, R. (2021). Research on routing and scheduling algorithms for the simultaneous transmission of diverse data streaming services on the industrial internet. *Scientific Reports*, *11*(1). https://doi.org/10.1038/s41598-021-97613-9

Song, Y., Guo, C., Xu, P., Li, L., & Zhang, R. (2021). Research on routing and scheduling algorithms for the simultaneous transmission of diverse data streaming services on the industrial internet. *Scientific Reports*, *11*(1), 18351. https://doi.org/10.1038/s41598-021-97613-9

Talaat, F. M., Saleh, A. I., Ali, H. A., & Ali, S. H. (2019). Effective Load Balancing Strategy (ELBS) for Real-Time Fog Computing Environment Using Fuzzy and Probabilistic Neural Networks. *Journal of Network and Systems Management*. https://doi.org/10.1007/s10922-019-09490-3

Taley, M. M., & Keole, R. R. (2015). Study of Load Balancing In Distributed Computing Environment.

Tanenbaum, A., Wetherall, D., & Feamster, N. (2021). *Computer Networks, EBook, Global Edition* (Sixth edition, Global Edition. ed.). Pearson Education, Limited.

Tawfeeg, T. M., Yousif, A., Hassan, A., Alqhtani, S. M., Hamza, R., Bashir, M. B., & Ali, A. (2022). Cloud Dynamic Load Balancing and Reactive Fault Tolerance Techniques: A Systematic Literature Review (SLR). *IEEE Access*, *10*. https://doi.org/10.1109/ACCESS.2022.3188645

Thiele, D., & Ernst, R. (2016). Formal worst-case performance analysis of time-sensitive Ethernet with frame preemption. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1-9). IEEE. https://doi.org/10.1109/ETFA.2016.7733740

Todorov, D., Valchanov, H., & Aleksieva, V. (2020). Load Balancing model based on Machine Learning and Segment Routing in SDN. In *2020 International Conference Automatics*

and Informatics (ICAI) (pp. 1-4). IEEE. https://doi.org/10.1109/ICAI50593.2020.9311385

Vlk, M., Hanzalek, Z., Brejchova, K., Tang, S., Bhattacharjee, S., & Fu, S. (2020). Enhancing Schedulability and Throughput of Time-Triggered Traffic in IEEE 802.1Qbv Time-Sensitive Networks. *IEEE Transactions on Communications*, *68*(11). https://doi.org/10.1109/TCOMM.2020.3014105

Wan, C.-Y., Eisenman, S. B., & Campbell, A. T. (2011). Energy-efficient congestion detection and avoidance in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, *7*(4), 1-31. https://doi.org/10.1145/1921621.1921626

Wang, H., Xie, H., Qiu, L., Yang, Y. R., Zhang, Y., & Greenberg, A. (2006). COPE traffic engineering in dynamic networks. *ACM SIGCOMM Computer Communication Review*, *36*(4), 99. https://doi.org/10.1145/1151659.1159926

Wang, Y.-T., & Hung, K.-M. (2012). Fuzzy logic based neural network models for load balancing in wireless networks. *Journal of communications and networks*, *10*(1), 38-43.

Wang, Y., Chen, J., Ning, W., Yu, H., Lin, S., Wang, Z., . . . Chen, C. (2021). A time-sensitive network scheduling algorithm based on improved ant colony optimization. *Alexandria Engineering Journal*, *60*(1), 107-114. https://doi.org/10.1016/j.aej.2020.06.013

Wehrle, K., Günes, M., & Gross, J. (2010). *Modeling and tools for network simulation*. Springer. https://doi.org/10.1007/978-3-642-12331-3

Wei, W., Yi, S., Kai, Z., Kaafar, M. A., Dan, L., & Zhongcheng, L. (2014). Freeway: Adaptively Isolating the Elephant and Mice Flows on Different Transmission Paths. In *2014 IEEE 22nd International Conference on Network Protocols* (pp. 362-367). IEEE. https://doi.org/10.1109/ICNP.2014.59

Weichlein, T., Zhang, S., Li, P., & Zhang, X. (2023). Data Flow Control for Network Load Balancing in IEEE Time-Sensitive Networks for Automation. *IEEE Access*.

Wilson, S. P., & Deepalakshmi, P. (2019). DServ-LB: Dynamic server load balancing algorithm. *International Journal of Communication Systems*, *32*(1). https://doi.org/10.1002/dac.3840

Wisniewski, L., Hameed, M., Schriegel, S., & Jasperneite, J. (2009). A Survey of Ethernet Redundancy Methods for Real-Time Ethernet Networks and its Possible Improvements. *IFAC Proceedings Volumes*, *42*(3), 163-170. https://doi.org/10.3182/20090520-3-KR-3006.00024

Wollschlaeger, M., Sauter, T., & Jasperneite, J. (2017). The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine*, *11*(1). https://doi.org/10.1109/MIE.2017.2649104

Wu, G.-L., & Mark, J. (1993). Discrete time analysis of leaky-bucket congestion control. *COMPUTER NETWORKS AND ISDN SYSTEMS*, *26*(1), 79.

Wu, H., Ren, F., Mu, D., & Gong, X. (2009). An efficient and fair explicit congestion control protocol for high bandwidth-delay product networks. *Computer Communications*, *32*(7), 1138-1147. https://doi.org/10.1016/j.comcom.2008.11.016

Xiao, Y., Zhu, K., & Choo Liaw, H. (2005). Generalized synchronization control of multi-axis motion systems. *Control Engineering Practice*, *13*(7), 809-819. https://doi.org/10.1016/j.conengprac.2004.09.005

Yang, W., Jidong, C., Wei, N., Hao, Y., Shimei, L., Zhidong, W., . . . Chao, C. (2021). A time-sensitive network scheduling algorithm based on improved ant colony optimization. *Alexandria Engineering Journal*, *60*(1), 107-114.

Yao, S., Gao, G., & Gao, Z. (2021). On multi-axis motion synchronization: The cascade control structure and integrated SMC-ADRC design. *ISA transactions*, *109*, 259-268. https://doi.org/10.1016/j.isatra.2020.10.012

Yin, N., & Hluchyj, M. (1994). On Closed-Loop Rate Control for ATM Cell Relay Networks. In *5th Annual joint conference on computer communications*

(pp. 99-109). IEEE; 1994.

Yuen, W. K., Yeung, K. H., & Yan, F. (2011). Fast RSTP Convergence By Using Backup VLANs. In (pp. 338-338). WSEAS; 2011.

Zaki, M. J., Wei, L., & Parthasarathy, S. (1996, 6-9 Aug. 1996). Customized dynamic load balancing for a network of workstations. Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing,

Zhang, J., Yu, F. R., Wang, S., Huang, T., Liu, Z., & Liu, Y. (2018). Load Balancing in Data Center Networks: A Survey. *IEEE Communications Surveys & Tutorials*, *20*(3). https://doi.org/10.1109/COMST.2018.2816042

Zhang, Z., & Zhang, X. (2010). A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In *2010 The 2nd International Conference on Industrial Mechatronics and Automation* (pp. 240-243). https://doi.org/10.1109/ICINDMA.2010.5538385

Zhao, L., Pop, P., & Steinhorst, S. (2022). Quantitative Performance Comparison of Various Traffic Shapers in Time-Sensitive Networking. *IEEE Transactions on Network and Service Management*, *19*(3). https://doi.org/10.1109/TNSM.2022.3180160

Zhou, Z., Yan, Y., Berger, M., & Ruepp, S. (2018). Analysis and modeling of asynchronous traffic shaping in time sensitive networks. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)* (pp. 1-4). IEEE. https://doi.org/10.1109/WFCS.2018.8402376

Appendix 1: Load Calculations of Seamless Communication Use Case

The following tables show the detailed calculation of the load for the example of seamless M2M communication as referred to in Section 7.4. The numbers in the cells represent used bandwidth in per cent.

A symmetric load distribution is the result, if the traffic between two stations is equal in both directions as shown in Table 0.1. Each link has equal load sums of 21 per cent.

Table 0.1: Example of effects of seamless symmetric traffic contribution

Source->Sink/ Direction	Link/Direction							
	L1 /c	L1 /cc	L2 /c	L2 /cc	L3 /c	L3 /cc	L4 /c	L4 /cc
AC1->AC2 /c	5							
AC1->AC2 /cc				5		5		5
AC1->AC3 /c	5		5					
AC1->AC3 /cc						5		5
AC1->AC4 /c	5		5		5			
AC1->AC4 /cc								5
AC2->AC1 /c			5		5		5	
AC2->AC1 /cc		5						
AC2->AC3 /c			2					
AC2->AC3 /cc		2				2		2
AC2->AC4 /c			2		2			
AC2->AC4 /cc		2						2
AC3->AC1 /c					5		5	
AC3->AC1 /cc		5		5				
AC3->AC2 /c	2				2		2	
AC3->AC2 /cc				2				
AC3->AC4 /c					2			
AC3->AC4 /cc		2		2				2
AC4->AC1 /c							5	
AC4->AC1 /cc		5		5		5		

AC4->AC2 /c	2						2	
AC4->AC2 /cc				2		2		
AC4->AC3 /c	2		2				2	
AC4->AC3 /cc						2		
Sum	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**
c: clockwise								
cc: counterclockwise								

The asymmetric load distribution is the result, if the traffic between two stations is not equal in both directions as shown in Table 0.2. The link loads for the example differ between 15 and 21 per cent.

Table 0.2: Example of effects of seamless asymmetric traffic contribution

Source->Sink/ Direction	Link/Direction							
	L1 /c	L1 /cc	L2 /c	L2 /cc	L3 /c	L3 /cc	L4 /c	L4 /cc
AC1->AC2 /c	5							
AC1->AC2 /cc				5		5		5
AC1->AC3 /c	5		5					
AC1->AC3 /cc						5		5
AC1->AC4 /c	5		5		5			
AC1->AC4 /cc								5
AC2->AC1 /c			3		3		3	
AC2->AC1 /cc		3						
AC2->AC3 /c			2					
AC2->AC3 /cc		2				2		2
AC2->AC4 /c			2		2			
AC2->AC4 /cc		2						2
AC3->AC1 /c					3		3	
AC3->AC1 /cc		3		3				
AC3->AC2 /c	2				2		2	
AC3->AC2 /cc				2				
AC3->AC4 /c					2			
AC3->AC4 /cc		2		2				2
AC4->AC1 /c							3	
AC4->AC1 /cc		3		3		3		
AC4->AC2 /c	2						2	
AC4->AC2 /cc				2		2		
AC4->AC3 /c	2		2				2	
AC4->AC3 /cc						2		
Sum	**21**	**15**	**19**	**17**	**17**	**19**	**15**	**21**
c: clockwise								
cc: counterclockwise								

Appendix 2: Ns-3 Simulation Code

This chapter provides supplementary documentation of the ns-3 network simulation setup for the validation use cases of Section 6.7. A detailed code print of the ns-3 C++ code follows. The structure of the data frames and most important data structures used for communication therein are shown in Figure 0.1. For the UML description of the modules refer to Section 6.7.

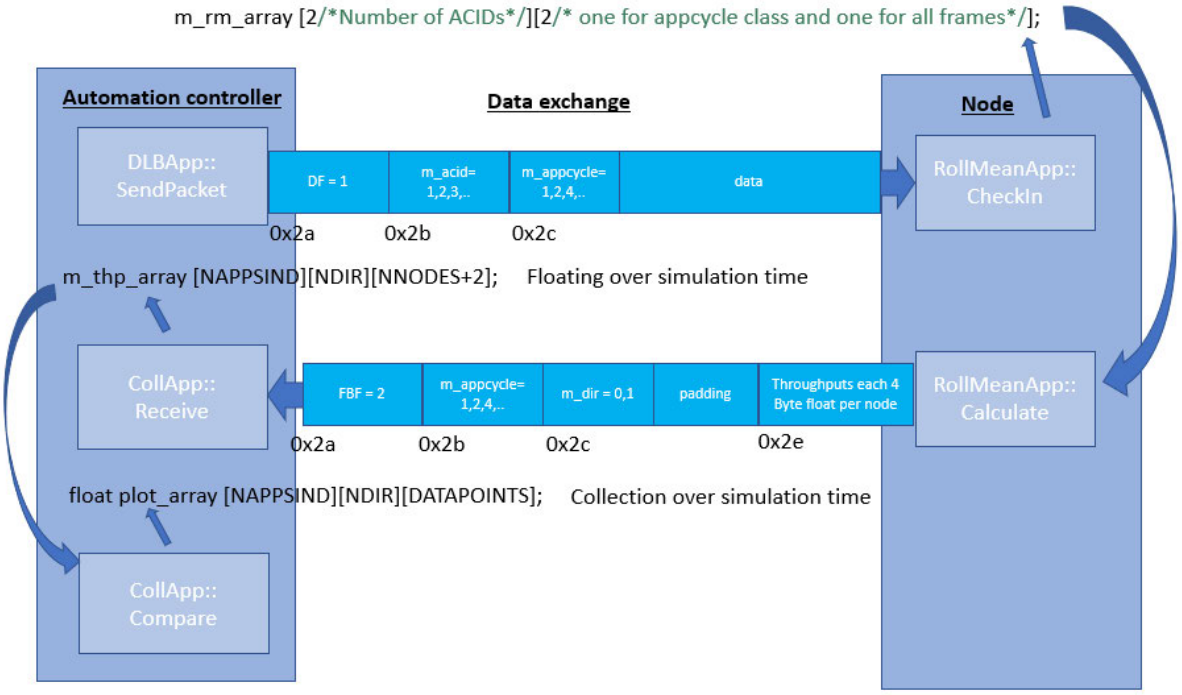


Figure 0.1: The structure of the data frames and most important data structures for the simulation

Code section:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
// This ns-3 simulation supplements the PhD thesis:
// A method for optimum control of dynamic load distribution in time-sensitive communication networks for manufacturing automation
//
// Author: Thomas Weichlein, University of Gloucestershire, UK
//
// Details:
// Network topology consists of a ring topology with 1 AC and 9 devices including integrated two port bridges (n1 to n9).
//
// This version supports the use cases (UC) according to thesis: UC1, UC2, UC3, UC4, UC5, UC6.1 .2 .3 .4, UC7,
//
// - UDP stream as Control data 1,2,4 and 8 ms from n0 (AC 10.1.1.0) to n9 (10.1.1.18) and from n0 (AC 10.1.1.20) to n1 (10.1.1.2)
// - UDP stream as interference Control data 1 ms and 4 ms from n2 (PLC 10.1.1.4) to n4 (10.1.1.8)
// - UDP stream as interference Control data 2 ms and 8 ms from n1 (PLC 10.1.1.2) to n5 (10.1.1.10)
// - Separate Flow Controllers for each application communication cycle class of 1 ms, 2 ms, 4 ms, and 8 ms,
// - DropTail queues
// - Tracing of queues and packet receptions to file "LDCv011.tr"
// - Creation of plot files 2D and 3D
//
// General hints:
// - The start of the controllers must be adapted to the integration time of the rolling mean measurement, that is must be later, to
avoid start jumps
// e.g. 1 ms Int. Time : 0.01 (10 ms) start delay; 8 ms Int. Time : 0.02 (20 ms) start delay;
//
#include <iostream>
#include <fstream>
```

```
#include <vector>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/bridge-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/gnuplot.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("LDCSim");
// Load Control is off?
//#define NOCONTROL
// Common Load Control for all app cycles?
//#define COMMONCONTROL
//If Load Control is on, include Distribution Controller?
//#define DISTCTRL
//switch on interferences?
#define INTERFERENCE_1MS
#define INTERFERENCE_2MS
#define INTERFERENCE_4MS
#define INTERFERENCE_8MS
// kind of plot
#define PLOT_2D
//#define PLOT_3D
#ifdef DISTCTRL
#define FLOW_1MS_IN LDCcw1msApp->m_flowctrlin
#else
#define FLOW_1MS_IN CollAC1App->m_thp_array[ALLAPPSIND][CW][NNODES+1]
```

```
#endif
//global simulation constants and variables:
//general
#define TAS_WINDOW_NS                   200000       //Size of the TAS or EST Window in ns
#define DATARATE                        1000000000   //data rate in bit per second
#define MIN_PACKET_SIZE                 242          //for this simulation, otherwise 96 is minimum
#define SIMSTART                        0.0          //Time of general start of simulation
#define SIMEND                          0.4          //Time of general end of simulation
#define LDC_ENDDELAY                    0.008        //Delay to continue with throughput rolling mean measurement
#define CTRL_SIMSTART                   0.015        //Time of control start of simulation
#define RM_1MS_DELAY                    0.0008       //Delay to place the start of rolling mean measurement
#define RM_2MS_DELAY                    0.0007       //Delay to place the start of rolling mean measurement
#define RM_4MS_DELAY                    0.0006       //Delay to place the start of rolling mean measurement
#define RM_8MS_DELAY                    0.00055      //Delay to place the start of rolling mean measurement
#define RM_ENDDELAY                     0.0002       //Delay to continue with throughput rolling mean measurement
#define COLL_STARTDELAY                 0.00062      //Delay to start with throughput feedback collections
#define COLL_ENDDELAY                   0.008        //Delay to continue with throughput feedback collections
#define CTRLRAMPSTEP                    4           //Ramp of reference to avoid switch-on-jump
#define DIST_STARTDELAY                 0.00064      //Delay to start with distribution controller
#define DIST_ENDDELAY                   0.00         //Delay to continue with distribution controller
#define FLOW_STARTDELAY                 0.00068      //Delay to start with flow controller
#define FLOW_ENDDELAY                   0.001        //Delay to continue with flow controller
#define CTRL_STARTDELAY_1               660000       //Offset to start with flow controller in ns for seamless interaction
#define CTRL_STARTDELAY_2               700000       //Second offset to start with flow controller in ns for seamless interaction
#define INTERFERENCE_START_1MS          0.05         //time to start with the 1 ms interference load step
#define INTERFERENCE_START_2MS          0.15         //time to start with the 2 ms interference load step
#define INTERFERENCE_START_4MS          0.25         //time to start with the 4 ms interference load step
#define INTERFERENCE_START_8MS          0.35         //time to start with the 8 ms interference load step
#define INTERFERENCE_STOP_1MS           0.4          //time to stop with the 1 ms interference load step
```

```
//frames
#define DF                1            //frame ID data frame
#define FBF               2            //frame ID throughput feedback frame
#define UDP               0x11         //UDP frame
enum NODEID         {N0,          //Node ID node 0 which is always AC1
                     N1,          //Node ID node 1
                     N2,          //Node ID node 2
                     N3,          //Node ID node 3 or AC2 depending on use case simulation
                     N4,          //Node ID node 4
                     N5,          //Node ID node 5 or ACC3 depending on use case simulation
                     N6,          //Node ID node 6
                     N7,          //Node ID node 7
                     N8,          //Node ID node 8 or ACC4 depending on use case simulation
                     N9,          //Node ID node 9
                     NNODES       //Number of Nodes
};
//rolling mean calculation
#define RM_WINDOWSIZE                   10000  //for now for 1ms, otherwise later calculate
:5*32*TAS_WINDOW_NS/((1*8*MIN_PACKET_SIZE))    //The maximum window size for the rolling mean calculation in number of packets
#define APPSPECIFIC                     0      //access application cycle specific throughput measurement
#define APPSCOLLECTIVE                  1      //access all application cycle throughput measurement
enum APPINDEX {
       APP1MSIND,
       APP2MSIND,
       APP4MSIND,
       APP8MSIND,
       APP16MSIND,
       APP32MSIND,
       ALLAPPSIND, //for measurement by rolling mean measurement at node over all application cycles
```

```
       SUMAPPSIND, //for summing up at collection app over all application cycles
       NAPPSIND};  //to address the application cycle in arrays
enum APPCYCLE {
       APP1MS = 1,
       APP2MS,
       APP4MS = 4,
       APP8MS = 8,
       APP16MS = 16,
       APP32MS = 32,
       ALLAPPS = 0xff
       };  //to configure the application cycle
enum INTTIME {
       INT1MS = 1,
       INT2MS,
       INT4MS = 4,
       INT8MS = 8,
       INT16MS = 16,
       INT32MS = 32
};  //to code the application cycle
//distribution control and flow control
enum DIRECTION {CW,              //Clockwise direction for maximum calculation
                          CCW,          //Counterlockwise direction for maximum calculation
                          NDIR};        //Number of Directions
//plotting
#define DATAPOINTS                      400                 //maximum number of data points for the plot file
#define DATASTEP                        1                   //plot only every DATASTEP data point
//Dynamic load Balancing specific classes
//++++++++++++++++++++++++++++++++++++++
//A LDCApp sends application frames in one direction of the ring and provides control facilities
```

```
class LDCApp : public Application
{
public:
       static TypeId GetTypeId (void)
         {
           static TypeId tid = TypeId ("LDCApp")
             .SetParent (Object::GetTypeId ())
             .SetGroupName ("MyGroup")
             .AddConstructor<LDCApp> ()
             .AddTraceSource ("NPackets",
                              "Number of Packets to trace.",
                              MakeTraceSourceAccessor (&LDCApp::NPackets),
                              "ns3::TracedValueCallback::Int32")
             ;
           return tid;
         }
        LDCApp ();
        virtual ~LDCApp();
        void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t nPackets, DataRate dataRate, uint32_t
appcycle, uint32_t acid, bool externalref, float* extref, float* diffthp);
        uint32_t plot_array [4][DATAPOINTS];
        TracedValue<int32_t> NPackets;
        float*              m_diffthroughput;               //difference of throughput
        float               m_distctrlin;                   //input for distribution control
        float               m_distctrlout;                  //output from distribution control, input for flow control
        float               m_flowctrlin;                   //flow control input
        float               m_flowctrlout;                  //flow control output, to be translated into m_nPackets for send unit
private:
        virtual void StartApplication (void);
```

```
        virtual void StopApplication (void);
        void ScheduleTx (void);
        void ScheduleTxCycle (void);
        void SendPacket (void);
        void ReceivePacket (Ptr<Socket> socket);
        void Control (void);
        Ptr<Packet>     m_packet;
        Ptr<Socket>     m_socket;
        Address         m_peer;
        uint32_t        m_packetSize;
        uint32_t        m_nPackets;
        int32_t         m_deltaPackets;
        DataRate        m_dataRate;
        EventId         m_sendEvent;
        bool            m_running;
        uint32_t        m_packetsSent;
        uint32_t        m_appcycle;
        uint32_t        m_appcycle_ind;
        uint32_t        m_acid;
        uint32_t        m_totalsent;
        uint32_t        m_j;                          //iterator for rolling mean
        double          m_throughputs [2][NNODES];    //received throughput measurements from nodes
        double          m_maxthroughput_cw;           //maximum throughput clockwise direction
        double          m_maxthroughput_ccw;          //maximum throughput countrclockwise direction
        bool            m_externalref;                //use reference from coupled LDC
        float*          m_extref;                     //reference from coupled LDC flow controller output
//  Gnuplot2dDataset  m_2ddataset;
//  Gnuplot3dDataset  m_3ddataset;
};
```

```
LDCApp::LDCApp ()
  : m_diffthroughput(0),
       m_distctrlin (0),
       m_distctrlout (0),
       m_flowctrlin (0),
       m_flowctrlout (0),
       m_packet (0),
       m_socket (0),
       m_peer (),
       m_packetSize (0),
       m_nPackets (0),
       m_deltaPackets (0),
       m_dataRate (0),
       m_sendEvent (),
       m_running (false),
       m_packetsSent (0),
       m_appcycle (1),
       m_appcycle_ind (0),
       m_acid (1),
       m_totalsent (0),
       m_j (0),
//     m_throughputs (0),
       m_maxthroughput_cw (0),
       m_maxthroughput_ccw (0),
       m_externalref (false),
       m_extref (0)
//     m_2ddataset (),
//     m_3ddataset ()
{
```

```
}
LDCApp::~LDCApp()
{
  m_socket = 0;
}
void
LDCApp::Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t nPackets, DataRate dataRate, uint32_t appcycle,
uint32_t acid,  bool externalref, float* extref, float* diffthp)
{
  m_socket = socket;
  m_peer = address;
  m_packetSize = packetSize;
  m_nPackets = nPackets;
  m_dataRate = dataRate;
  m_appcycle = appcycle;
  m_acid = acid;
  m_externalref = externalref;
  m_extref = extref;
  m_diffthroughput = diffthp;
}
void
LDCApp::StartApplication (void)
{
  m_running = true;
  m_packetsSent = 0;
  m_socket->Bind ();
  m_socket->Connect (m_peer);
  uint8_t buffer[3];
  buffer[0]= (uint8_t) DF; //data frame
```

```
  buffer[1]= (uint8_t) m_acid; //automation controller id for packet
  buffer[2]= (uint8_t) m_appcycle; //application cycle in ms for packet
  uint8_t* buf = buffer;
  m_packet = Create<Packet> (buf, m_packetSize);
  SendPacket ();
}
void
LDCApp::StopApplication (void)
{
  m_running = false;
  NS_LOG_INFO ("LDCApp: Total packets sent: " << m_totalsent);
  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }
  if (m_socket)
    {
      m_socket->Close ();
    }
}
void
LDCApp::SendPacket (void)
{
//  Ptr<Packet> packet = Create<Packet> (m_packetSize);
  m_socket->Send (m_packet);
  m_totalsent++;
  //to test trace:
  //++m_nPackets;
  //NPackets = m_nPackets;
```

```
  //  //this->m_2ddataset.Add (static_cast<double> (Simulator::Now ()), m_nPackets);
  //if (m_nPackets == 5)
  //     m_nPackets = 2;
  // NS_LOG_INFO ("Sending App: = " << this);
  if (++m_packetsSent < m_nPackets + m_deltaPackets)
    {
      ScheduleTx ();
    }
  else
  {
        ScheduleTxCycle ();
  }
}
//Schedule sending within the communication cycle
void
LDCApp::ScheduleTx (void)
{
  if (m_running)
    {
        if (m_packetsSent == 1)
        {
      //Schedule value update and convertion before and after flow controller
      Time tNextControl_1 (CTRL_STARTDELAY_1);
      m_sendEvent = Simulator::Schedule (tNextControl_1, &LDCApp::Control, this);
      Time tNextControl_2 (CTRL_STARTDELAY_2);
      m_sendEvent = Simulator::Schedule (tNextControl_2, &LDCApp::Control, this);
        }
      //Schedule next frame
      Time tNext (Seconds (m_packetSize * 8 / static_cast<double> (m_dataRate.GetBitRate ())));
```

```
      m_sendEvent = Simulator::Schedule (tNext, &LDCApp::SendPacket, this);
         //NS_LOG_INFO ("ScheduleTx in " << tNext);
    }
}
//Schedule sending for the application cycle
void
LDCApp::ScheduleTxCycle (void)
{
  if (m_running)
    {
      Time tNextCycle (Seconds ((m_appcycle/static_cast<double> (1000))- m_packetsSent * (m_packetSize * 8 / static_cast<double>
(m_dataRate.GetBitRate ())) ) );
      m_sendEvent = Simulator::Schedule (tNextCycle, &LDCApp::SendPacket, this);
         //NS_LOG_INFO ("ScheduleTxCycle in " << tNextCycle);
    }
  plot_array[0][m_j] = m_appcycle *(m_j+1);
  plot_array[1][m_j] = m_nPackets;
  if (m_j <= DATAPOINTS) m_j++;
  m_packetsSent = 0;
}
//Handle references and outputs, make LDC coupling, convert flow controller output into number of packets
void
LDCApp::Control (void)
{
       if (!m_externalref)
           {
#ifdef DISTCTRL
             m_distctrlin = (*m_diffthroughput)* (float)(-1);
             m_flowctrlin = m_distctrlout - (*m_diffthroughput);
```

```
//         m_flowctrlin = m_distctrlout - ((*m_diffthroughput) * (float)(-1));
#else
           m_flowctrlin = (*m_diffthroughput)* (float)(-1);
#endif
              NS_LOG_INFO ("diffthroughput is " << (*m_diffthroughput));
           }
       else
           {
           //Coupled to other LDC on node. No own distribution or flow control. Take result from other leading flow controller
output
              m_flowctrlout = *m_extref * (float)(-1);
           }
       //simulate with or without load control
#ifdef NOCONTROL
       m_deltaPackets = 0;
#else
       //Calculate packets and change algebraic sign as a positive difference means a reduction for this direction
       //     m_deltaPackets = (m_flowctrlout * (float)(-1))/(0.0001 * m_packetSize * 8);
       m_deltaPackets = m_flowctrlout/(0.0001 * m_packetSize * 8);
       //a controller can only compensate load differences within the border of its own introduced load
       if (m_deltaPackets > (int32_t)m_nPackets)
              {
                     m_deltaPackets = m_nPackets -3;
              }
       if (m_deltaPackets <= ((int32_t)m_nPackets * (-1)))
              {
                     m_deltaPackets = (m_nPackets * (-1)) + 3;
              }
       NS_LOG_INFO ("delta packets is " << m_deltaPackets);
```

```
#endif
}
#if 0
static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}
//static void
//RxDrop (Ptr<const Packet> p)
//{
//  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
//}
void
 IntTrace (int32_t oldValue, int32_t newValue)
 {
   std::cout << "Traced Test " << oldValue << " to " << newValue << std::endl;
 }
#endif
// The RollMeanApp builds the rolling mean measurement of throughput on a node (bridged device)
// For each app cycle class there will be one necessary as they finally will use different integration times.
// But have with "getall" also the possibility to capture all app cycle class frames for the common control use case which is to
improve.
class RollMeanApp : public Application
{
public:
       static TypeId GetTypeId (void)
         {
           static TypeId tid = TypeId ("RollMeanApp")
```

```
            .SetParent (Object::GetTypeId ())
            .SetGroupName ("MyGroup")
            .AddConstructor<RollMeanApp> ()
              ;
          return tid;
        }
  RollMeanApp ();
  virtual ~RollMeanApp();
  void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, DataRate dataRate, uint32_t direction, uint32_t appcycle,
uint32_t inttime, uint32_t node, bool getall);
  void CheckInPacket (ns3::Ptr<const ns3::Packet>);
  float      p_throughput[2][DATAPOINTS]; // in percent, one array for app cycle class, one for all app cycle class measurement
private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);
  void Calculate (void);
  Ptr<Packet>     m_packet;
  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint8_t         m_buffer [4 + 10 * sizeof(float)];
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  uint32_t        m_windowSize;  //number of measurement values in the storage window
  uint32_t        m_direction;   //clockwise or counterclockwise
  uint32_t        m_appcycle;    //app cycle class to capture
  uint32_t        m_inttime;     // integration time for rolling mean measurement
  bool            m_getall;      //if true the throughput is measured over all application cycles
```

```
  uint32_t        m_node;        //node number in the ring
  uint32_t        m_j;           // check in iterator for explicit app cycle
  uint32_t        m_k;           // check in iterator for all app cycle
  struct          rm_array_t {uint32_t nbytes[RM_WINDOWSIZE]; Time timestamp[RM_WINDOWSIZE];};
  rm_array_t      m_rm_array [2/*Number of ACIDs*/][2/* one for appcycle class and one for all frames*/];
  bool            m_running;
  uint32_t        m_totalcheckedin;
  uint32_t        m_totalallcheckedin;
  Time            m_currenttime;
  Time            m_windowstart;
  Time            m_windowupperpart;
  Time            m_windowstarttime;
  Time            m_arraystarttime;
  Time            m_simulatorlast;
  uint32_t        m_appcycle_ind;
  EventId         m_calcEvent;
  uint32_t        m_datapoint;
  uint32_t        m_i;             //loopcounter for debugging
  uint32_t        m_bytes;
  uint32_t        m_prevsize;
};
RollMeanApp::RollMeanApp ()
:      m_packet (0),
       m_socket (0),
       m_peer (),
       m_packetSize (0),
       m_nPackets (0),
       m_dataRate (0),
       m_sendEvent (),
```

```
      m_windowSize (RM_WINDOWSIZE),
    m_direction (0),
      m_appcycle (1),
      m_inttime (0),
      m_getall (false),
      m_node (0),
      m_j (0),
      m_k (0),
      m_running (false),
      m_totalcheckedin (0),
      m_totalallcheckedin (0),
      m_currenttime (0),
      m_windowstart (0),
      m_windowupperpart (0),
      m_windowstarttime (0),
      m_arraystarttime (0),
      m_simulatorlast (0),
      m_appcycle_ind (0),
      m_calcEvent (),
      m_datapoint (),
      m_i (0),
      m_bytes (0),
      m_prevsize (0)
{
}
RollMeanApp::~RollMeanApp()
{
}
//so far not needed:
```

```
void
RollMeanApp::Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, DataRate dataRate, uint32_t direction, uint32_t
appcycle, uint32_t inttime, uint32_t node, bool getall)
{
  m_socket = socket;
  m_peer = address;
  m_packetSize = packetSize;
  m_dataRate = dataRate;
  m_direction = direction;
  m_appcycle = appcycle;
  m_inttime = inttime;
  m_node = node;
  m_getall = getall;
}
void
RollMeanApp::StartApplication (void)
{
  m_running = true;
  m_socket->Bind ();
  m_socket->Connect (m_peer);
  m_buffer[0]= (uint8_t) FBF; //Feedback frame
  m_buffer[1]= (uint8_t) m_appcycle; //application cycle in ms for packet
  m_buffer[2]= (uint8_t) m_direction; //clockwise or conterclockwise
  uint8_t* buf = &m_buffer[0];
  m_packet = Create<Packet> (buf, m_packetSize);
  Time tNext (/*m_inttime **/ 1000000/*Seconds ((uint32_t) 0.005)*/);  //First calculation
  m_calcEvent = Simulator::Schedule (tNext, &RollMeanApp::Calculate, this);
//  NS_LOG_INFO ("Calculate RM in " << tNext);
//  NS_LOG_INFO (Simulator::Now () << "   " << Simulator::Now().GetSeconds());
```

```
}
void
RollMeanApp::StopApplication (void)
{
  m_running = false;
  NS_LOG_INFO ("Total app specific packets checked in: " << m_totalcheckedin);
  NS_LOG_INFO ("Total all apps packets checked in: " << m_totalallcheckedin);
  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }
  if (m_socket)
    {
      m_socket->Close ();
    }
#if (0)
  for (uint32_t i=0; i < 50; i++) //Check just a few entries visually
  {
         NS_LOG_INFO ("rm_array entry number: " << i);
         NS_LOG_INFO ("rm_array bytes = " << m_rm_array[0][0].nbytes[i]);
         NS_LOG_INFO ("rm_array timestamp = " << m_rm_array[0][0].timestamp[i] << std::endl);
  }
#endif
}
// Calculate the rolling means of all application cycles and VLANs
void
RollMeanApp::Calculate (void)
{
       // one calculation for app cycle class and one for all app cycles
```

```
for (uint32_t c = 0; c < 2; ++c)
{
m_currenttime = (Simulator::Now ());
m_arraystarttime = m_rm_array[0][c].timestamp[0];
m_bytes = 0;
m_i = 0;
uint32_t ind = 0;
if (c==0)
{
       ind = m_j; //app cycle specific calculation
}
if (c==1)
{
       ind = m_k; //calculation over all app cycles
}
if ((m_currenttime- (Time)m_inttime * 1000000) >= m_arraystarttime) //window is fully within array
       {
       m_windowstarttime = m_currenttime - (Time)m_inttime * 1000000;
       for (uint32_t i = ind; m_rm_array[0][c].timestamp[i] > m_windowstarttime; --i)
                  {
                     m_bytes = m_bytes + m_rm_array[0][c].nbytes[i];
                     m_i++;
                  }
         NS_LOG_INFO ("RollMeanApp: Sum of " << m_i << "packets");
       }
else //window suffered a turnover within array or has just started
       {
       for (int32_t i = ind; i >= 0; --i)
                              {
```

```
                                m_bytes = m_bytes + m_rm_array[0][c].nbytes[i];
                                m_i++;
                              }
            if (m_rm_array[0][c].timestamp[RM_WINDOWSIZE-1] > (Time)(0)) // turnover
            {
            m_windowupperpart = m_currenttime - m_rm_array[0][c].timestamp[RM_WINDOWSIZE-1];
            m_windowstarttime = m_rm_array[0][c].timestamp[RM_WINDOWSIZE-1] - ((Time)(m_inttime * 1000000) - m_windowupperpart);
            for (uint32_t i = RM_WINDOWSIZE-1; m_rm_array[0][c].timestamp[i] >= m_windowstarttime; --i)
                              {
                                m_bytes = m_bytes + m_rm_array[0][c].nbytes[i];
                                m_i++;
                              }
            }
            NS_LOG_INFO ("RollMeanApp: Sum of " << m_i << "packets");
            }
            p_throughput [c][m_datapoint]= ((float)m_bytes * 8)/(m_inttime * 10000)/* 5 * 1000000 ns per ms divided by 100 is 1
per cent*/;
            NS_LOG_INFO ("RollMeanApp: Throughput = " << p_throughput[c][m_datapoint] << " %" << std::endl);
       m_i = 0;
       }
            if(m_running)
            {
                    Time tNext (1000000); //next calculation (is not integration time but calc cycle
                    m_calcEvent = Simulator::Schedule (tNext, &RollMeanApp::Calculate, this);
//                  NS_LOG_INFO ("Calculate in " << tNext << std::endl);
            }
            //For now, send the throughput circle frame here directly to AC1, Later on only to the next node.
            //Later on also one frame per direction and app cycle which comes with different rolling mean apps.
            //*(((float*)(&m_buffer[4])) + m_node) = p_throughput [0][m_datapoint];
```

```
            // Send app cycle class throughput measurement
//            *((uint8_t*)(&m_packet[0x2b])) = (uint8_t)m_appcycle;
//            *(((float*)((uint8_t*)(&m_packet[0x2e]))) + m_node) = p_throughput[0] [m_datapoint];
//            *((uint8_t*)((&m_packet->m_buffer)+1)) = (uint8_t)m_appcycle;
//            *(((float*)((uint8_t*)(&m_packet->m_buffer[3]))) + m_node) = p_throughput[0] [m_datapoint];
            m_buffer[1] = (uint8_t) m_appcycle;
            *(((float*)(&m_buffer[4])) + m_node) = p_throughput [0] [m_datapoint];
            uint8_t* buf = &m_buffer[0];
            m_packet = Create<Packet> (buf, m_packetSize);
            m_socket->Send (m_packet);
//            NS_LOG_INFO ("Packet UID " << m_packet->GetUid());
//            NS_LOG_INFO ("Packet RefCnt " << m_packet->GetReferenceCount() << std::endl);
            // Send overall app cycle class throughput measurement out of the 8 ms app cycle measurement
            if (m_appcycle == APP8MS)
            {
            m_buffer[1] = ALLAPPS;
            *(((float*)(&m_buffer[4])) + m_node) = p_throughput [1][m_datapoint];
            m_packet = Create<Packet> (buf, m_packetSize);
            m_socket->Send (m_packet);
//            NS_LOG_INFO ("Packet UID " << m_packet->GetUid());
//            NS_LOG_INFO ("Packet RefCnt " << m_packet->GetReferenceCount() << std::endl);
            }
            if(m_datapoint < DATAPOINTS)
                  {
                        m_datapoint++;
                  }
}
void
RollMeanApp::CheckInPacket ( Ptr<const Packet> pPacket)
```

```
{
        uint8_t buffer [0x30];
        uint8_t* p_buf = buffer;
        Ptr<const Packet> ppacket = pPacket;
        ppacket->CopyData (p_buf, 0x30);
//      NS_LOG_INFO ("tx callback: pointer = " << *ppacket << std::endl << std::endl );
//      std::cout << "send callback: pointer = " << *ppacket;
//      NS_LOG_INFO ("tx callback: packet size = " << ppacket->GetSize());
//      NS_LOG_INFO ("Check In App: = " << this);
        if (m_running)
        {
        //   fetch only UDP  process data
            if (buffer[0x17] == UDP)
               {
               //fetch only data frames
               if (buffer[0x2a] == DF)
               {
                     //check into first buffer if of this explicit app cycle
                      if (buffer[0x2c] == m_appcycle)
                           {
                             if (m_j == m_windowSize -1)
                                    {
                                           m_j = 0;
//                                         NS_LOG_INFO ("iterator m_j turnover, set to " << m_j);
                                    }
                             else
                                    {
                                           m_j++;
                                    }
```

```
                            m_rm_array[(buffer[0x2b])-1/*ACID*/][0].nbytes[m_j] = ppacket->GetSize();
                            m_rm_array[(buffer[0x2b])-1/*ACID*/][0].timestamp[m_j] = (Simulator::Now ());
                            m_totalcheckedin++;
                          }
#if 0
              /*Check all frames into second buffer for common throughput measurement?*/
                      if ((buffer[0x2c] == APP1MS) || (buffer[0x2c] == APP2MS) || (buffer[0x2c] == APP4MS) || (buffer[0x2c] ==
APP8MS) || (buffer[0x2c] == APP16MS) || (buffer[0x2c] == APP32MS))
                                    {
                                      if (m_getall)
                                      {
                                           if (m_k == m_windowSize -1)
                                           {
                                           m_k = 0;
//                                         NS_LOG_INFO ("iterator m_k turnover, set to " << m_k);
                                           }
                                           else
                                           {
                                           m_k++;
                                           }
                                           m_rm_array[(buffer[0x2b])-1/*ACID*/][1].nbytes[m_k] = ppacket->GetSize();
                                           m_rm_array[(buffer[0x2b])-1/*ACID*/][1].timestamp[m_k] = (Simulator::Now ());
                                           m_totalallcheckedin++;
                                      }
                                    }
#endif
#if 0
                if (((ppacket->GetSize() != m_prevsize)&& (m_prevsize !=0))|| (ppacket->GetSize() == 0))
                {
```

```
                    NS_LOG_INFO ("Error packet size: previous size = " << m_prevsize << " ,current size is " << ppacket->GetSize()
<< std::endl);
                }
              m_prevsize = ppacket->GetSize();
#endif
#if 0
                NS_LOG_INFO ("rm_array bytes = " << m_rm_array[0][0].nbytes[m_j]);
                NS_LOG_INFO ("rm_array timestamp = " << m_rm_array[0][0].timestamp[m_j]);
                NS_LOG_INFO ("delta time = " << m_rm_array[0][0].timestamp[m_j] - m_simulatorlast);
                NS_LOG_INFO ("iterator m_j = " << m_j << std::endl);
                m_simulatorlast = Simulator::Now ();
#endif
                }
          }
          }
}
// The PIDCtrlApp implements the PID Controller
class PIDCtrlApp : public Application
{
public:
       static TypeId GetTypeId (void)
          {
            static TypeId tid = TypeId ("PIDCtrlApp")
              .SetParent (Object::GetTypeId ())
              .SetGroupName ("MyGroup")
              .AddConstructor<PIDCtrlApp> ()
                ;
            return tid;
          }
```

```
  PIDCtrlApp ();
  virtual ~PIDCtrlApp();
  void Setup (double kp, double ki, double kd, uint32_t intstep, double thres, float* ref, float* out, std::string type);
  float                      p_throughput[DATAPOINTS]; // in percent
private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);
  void Calculate (void);
  double              m_ref;            //input
  uint32_t            m_rampcnt;        //counter to avoid start-up-jumps by ramping up reference.
  Double              m_threshold;      //threshold to damp responsiveness
  double              m_kp;             //Proportional factor
  double              m_ki;             //Integral factor
  double              m_kd;             //Differential factor
  float               m_out;            //Output
  bool                m_running;
  EventId             m_calculateEvent;
  Double              m_int;            //integral sum up
  Double              m_lastint;        //last integral sum up
  Double              m_lasttime;       //point in time of last calculation
  uint32_t            m_intstep;        //integration time step in ns, that is calculation cycle for PID controller
  double              m_lastref;        //reference at last calculation
  float*              m_refptr;         //pointer to controller reference input
  float*              m_outptr;         //pointer to controller output
  std::string         m_type;           //type of PID (distribution or flow control)
};
PIDCtrlApp::PIDCtrlApp ()
  : m_ref (0),
       m_rampcnt (0),
```

```
      m_threshold (0),
      m_kp (0),
      m_ki (0),
      m_kd (0),
      m_out (0),
      m_running (false),
      m_calculateEvent (),
      m_int (0),
      m_lastint (0),
      m_lasttime (0),
      m_intstep (0),
      m_lastref (0),
      m_refptr (0),
      m_outptr (0),
      m_type ()
{
}
PIDCtrlApp::~PIDCtrlApp()
{
}
void
PIDCtrlApp::Setup (double kp, double ki, double kd, uint32_t intstep, double thres, float* ref, float* out, std::string type)
{
  m_kp = kp;
  m_ki = ki;
  m_kd = kd;
  m_intstep = intstep;
  m_threshold = thres;
  m_refptr = ref;
```

```
  m_outptr = out;
  m_type = type;
}
void
PIDCtrlApp::StartApplication (void)
{
  m_running = true;
  Time tNext (1000000/*m_intstep*/);
  m_calculateEvent = Simulator::Schedule (tNext, &PIDCtrlApp::Calculate, this);
//NS_LOG_INFO ("Calculate PIDCtrl in " << tNext);
}
void
PIDCtrlApp::StopApplication (void)
{
  m_running = false;
}
// Calculate the PID Controller
// used for distribution control and flow control
void
PIDCtrlApp::Calculate (void)
{
#if 1
       NS_LOG_INFO ("PID Type is " << m_type);
       NS_LOG_INFO ("Reference is " << *m_refptr);
       //avoid "switch-on-jumps": ramp up reference at control start.
       m_ref = *m_refptr;
       if (m_rampcnt == 0)
       {
              m_ref = m_ref/CTRLRAMPSTEP;
```

```
		m_rampcnt++;
	}
	else
	{
		m_ref = m_ref * (m_rampcnt/CTRLRAMPSTEP);
		if (m_rampcnt < CTRLRAMPSTEP)
		{
			m_rampcnt++;
		}
	}
	if (fabs(m_ref) > m_threshold)
	{
		m_int = m_lastint + (m_ki * (m_ref) * (Simulator::Now ().GetSeconds() - m_lasttime));
		*m_outptr = m_kp * m_ref + m_int + ((m_ref - m_lastref)/(Simulator::Now ().GetSeconds() - m_lasttime) * m_kd);
		NS_LOG_INFO ("Time is " << Simulator::Now ().GetSeconds());
		NS_LOG_INFO ("integral difference is " << (m_ki * m_ref * (Simulator::Now ().GetSeconds() - m_lasttime)));
		NS_LOG_INFO ("integral new sum is " << m_int);
		NS_LOG_INFO ("differential part is  " << ((m_ref - m_lastref)/(Simulator::Now ().GetSeconds() - m_lasttime) * m_kd));
		m_lastref = m_ref;
		m_lasttime = Simulator::Now().GetSeconds();
		m_lastint = m_int;
	}
	NS_LOG_INFO ("Output is " << *m_outptr << std::endl);
	if(m_running)
		{
			Time tNext (1000000/*m_intstep*/);
			m_calculateEvent = Simulator::Schedule (tNext, &PIDCtrlApp::Calculate, this);
//			NS_LOG_INFO ("Calculate PID in " << tNext);
		}
```

```
#endif
}
//A collector application receives the throughput feedbacks from the nodes, compares and provides
//the difference as input for the distribution control
class CollApp : public Application
{
public:
       static TypeId GetTypeId (void)
         {
           static TypeId tid = TypeId ("CollApp")
             .SetParent (Object::GetTypeId ())
             .SetGroupName ("MyGroup")
             .AddConstructor<CollApp> ();
#if 0
               .AddTraceSource ("NPackets",
                                "Number of Packets to trace.",
                                MakeTraceSourceAccessor (&CollApp::NPackets),
                                "ns3::TracedValueCallback::Int32")
             ;
#endif
           return tid;
         }
        CollApp ();
        virtual ~CollApp();
        void Setup (uint32_t node);
        void ReceivePacket (ns3::Ptr<const ns3::Packet>);
        void Compare (void);
         //array of throughputs per direction, app, and node. forelast element to hold maximum
         //last element to hold the difference to the other direction
```

```
        float               m_thp_array [NAPPSIND][NDIR][NNODES+2];
      float plot_array [NAPPSIND][NDIR][DATAPOINTS];
//      TracedValue<int32_t> NPackets;
private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);
  bool            m_running;
  EventId         m_compEvent;
  uint32_t        m_node;
  uint32_t        m_appcycle_ind;
  uint32_t        m_datapoint;
}
;
CollApp::CollApp ()
  : m_running (false),
       m_compEvent (),
       m_node (0),
       m_appcycle_ind (0),
       m_datapoint (0)
{
}
CollApp::~CollApp()
{
}
void
CollApp::Setup (uint32_t node)
{
  m_node = node;
}
```

```
void
CollApp::StartApplication (void)
{
  m_running = true;
  //initialize throughput matrix
  for (uint32_t k = APP1MSIND; k < NAPPSIND; ++k)
       {
              for (uint32_t j = CW; j < NDIR; ++j)
              {
                     for (uint32_t i = N0; i < NNODES + 2; ++i)
                            {
                            m_thp_array [k][j][i] = 0;
                            }
              }
       }
  //initialize plot array
  for (uint32_t k = APP1MSIND; k < NAPPSIND; ++k)
       {
              for (uint32_t j = CW; j < NDIR; ++j)
              {
                     for (uint32_t i = 0; i < DATAPOINTS; ++i)
                            {
                            plot_array [k][j][i] = 0;
                            }
              }
       }
  Time tNext (1000000); //10 ms after simulation start
  m_compEvent = Simulator::Schedule (tNext, &CollApp::Compare, this);
//  NS_LOG_INFO ("Compare throughputs " << tNext);
```

```
}
void
CollApp::StopApplication (void)
{
  m_running = false;
}
//Receive Callback routine for the reception of nodes throughput feedbacks
void
CollApp::ReceivePacket (ns3::Ptr<const ns3::Packet> pPacket)
{
  uint8_t buffer [110]; //need at least 42 Bytes plus Header 24 Bytes
  uint8_t* p_buf = buffer;
  Ptr<const Packet> ppacket = pPacket;
  ppacket->CopyData (p_buf, ppacket->GetSize());
//    NS_LOG_INFO ("size of float is = " << sizeof(float) << std::endl );
//    NS_LOG_INFO ("rx callback: pointer = " << *ppacket << std::endl << std::endl );
//    std::cout << "rx callback: pointer = " << *ppacket;
//    NS_LOG_INFO ("Check In App: = " << this);
  if (m_running)
  {
  //   fetch only UDP process data
      if (buffer[0x17] == UDP)
         {
         //analyse throughput feedback frames and copy throughputs into array.
         if (buffer[0x2a] == FBF)
         {
                switch (buffer[0x2b]/*APPID*/)
                              {
                              case APP1MS:
```

```
                                    m_appcycle_ind = APP1MSIND;
                                    break;
                              case APP2MS:
                                    m_appcycle_ind = APP2MSIND;
                                    break;
                              case APP4MS:
                                    m_appcycle_ind = APP4MSIND;
                                    break;
                              case APP8MS:
                                    m_appcycle_ind = APP8MSIND;
                                    break;
                              case APP16MS:
                                    m_appcycle_ind = APP16MSIND;
                                    break;
                              case APP32MS:
                                    m_appcycle_ind = APP32MSIND;
                                    break;
                              case ALLAPPS:
                                    m_appcycle_ind = ALLAPPSIND;
                                    break;
                              default:
                             break;
                            }
            for (uint32_t i = N0; i < NNODES; ++i)
                  {
                  m_thp_array [m_appcycle_ind][buffer[0x2c]][i] = *((float*) (&buffer [0x2e + (i * sizeof(float))]));
#if 1
                  NS_LOG_INFO ("+++++ CollApp: +++++ " );
                  uint32_t temp = *((uint32_t*)(&buffer[0x2b]));
```

```
                    NS_LOG_INFO ("App cycle " << temp << "ms");
                    temp = *((uint32_t*)(&buffer[0x2c]));
                    NS_LOG_INFO ("Direction " << *((uint32_t*)(&buffer[0x2c])));
                    NS_LOG_INFO ("throughput of node " << i << " :" << m_thp_array [m_appcycle_ind][*((uint8_t*)&buffer[0x2c])][i]);
                    NS_LOG_INFO ("rx callback: packet size = " << ppacket->GetSize() << std::endl);
#endif
                    }
         }
         }
  }
#if 0
  if (packet->GetSize () > 0)
        {
         NS_LOG_INFO ("Received packet ");
        }
#endif
/*For further extensions: Make a coll app including forwarding for ACs, and a similar only forwarding app for nodes.
Create two circling throughput collection frames cw and ccw. Create a pass-further method with parameter direction.
In AC 1 make a buffer turnover of a buffer containing IP addresses in succession (to know where to pass to)
*/
}
// Compare the nodes throughputs
void
CollApp::Compare (void)
{
//sum up over the single application cycles per node and direction to build the overall bandwidth consumption (throughput) for app
dedicated control.
#if 1
       float sum = 0;
```

```
	for (uint32_t i = N0; i < NNODES; ++i)
	{
		for (uint32_t j = CW; j < NDIR; ++j)
		{
			for (uint32_t k = APP1MSIND; k < (NAPPSIND-2); ++k)
				{
				sum = sum + m_thp_array [k][j][i];
				}
			m_thp_array [SUMAPPSIND][j][i] = sum;
#if 0
			NS_LOG_INFO ("+++++ CollApp: for Plot+++++ " );
			NS_LOG_INFO ("data point " << m_datapoint);
			NS_LOG_INFO ("Sum of Node " << i);
			NS_LOG_INFO ("for direction " << j);
			NS_LOG_INFO ("Sum" << " :" << sum);
#endif
			sum = 0;
		}
	}
#endif
	//find maximum of each direction and application cycle and all application cycles and build and store difference
#if 1
	float max=0;
	static bool  take = false;
	for (uint32_t k = APP1MSIND; k < (NAPPSIND - 0); ++k)
	{
		for (uint32_t j = CW; j < NDIR; ++j)
		{
			for (uint32_t i = N0; i < NNODES; ++i)
```

```
                        {
                        if (m_thp_array [k][j][i] > max)
                          {
                                max = m_thp_array [k][j][i];
                            }
                        }
                m_thp_array [k][j][NNODES] = max;
                if (m_datapoint < DATAPOINTS)
                {
                plot_array [k][j][m_datapoint] = max;
                }
                max = 0;
                m_thp_array [k][j][NNODES + 1] = 0;
        }
        // only compare if both directions throughputs are available already
        if ((m_thp_array [k][CW][NNODES] != 0) && (m_thp_array [k][CCW][NNODES] != 0))
        {
        m_thp_array [k][CW][NNODES + 1] = (m_thp_array [k][CW][NNODES] - m_thp_array [k][CCW][NNODES])/2;
        NS_LOG_INFO ("maximum clockwise " << " :" << m_thp_array [k][CW][NNODES]);
        NS_LOG_INFO ("maximum counterclockwise " << " :" << m_thp_array [k][CCW][NNODES]);
        }
}
if (take == true) //store only every 2nd sample as CollApp is processed twice per millisecond for fast reaction
{
        take = false;
        m_datapoint++;
}
else
{
```

```
              take = true;
       }
       if (m_running)
       {
       Time tNext (500000/*m_intstep/2*/);
       m_compEvent = Simulator::Schedule (tNext, &CollApp::Compare, this);
//     NS_LOG_INFO ("Compare throughputs " << tNext);
       }
#endif
}
//A TrafficApp only sends frames in one direction of the ring, usually to simulate traffic interference
class TrafficApp : public Application
{
public:
       static TypeId GetTypeId (void)
         {
           static TypeId tid = TypeId ("TrafficApp")
             .SetParent (Object::GetTypeId ())
             .SetGroupName ("MyGroup")
             .AddConstructor<TrafficApp> ()
             .AddTraceSource ("NPackets",
                              "Number of Packets to trace.",
                              MakeTraceSourceAccessor (&LDCApp::NPackets),
                              "ns3::TracedValueCallback::Int32")
             ;
           return tid;
         }
        TrafficApp ();
        virtual ~TrafficApp();
```

```
        void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t nPackets, DataRate dataRate, uint32_t
appcycle, uint32_t acid);
        TracedValue<int32_t> NPackets;
private:
        virtual void StartApplication (void);
        virtual void StopApplication (void);
        void ScheduleTx (void);
        void ScheduleTxCycle (void);
        void SendPacket (void);
        Ptr<Packet>     m_packet;
        Ptr<Socket>     m_socket;
        Address         m_peer;
        uint32_t        m_packetSize;
        uint32_t        m_nPackets;
        DataRate        m_dataRate;
        EventId         m_sendEvent;
        bool            m_running;
        uint32_t        m_packetsSent;
        uint32_t        m_appcycle;
        uint32_t        m_acid;
        uint32_t            m_totalsent;
//  Gnuplot2dDataset  m_2ddataset;
//  Gnuplot3dDataset  m_3ddataset;
};
TrafficApp::TrafficApp ()
  : m_packet (0),
    m_socket (0),
    m_peer (),
    m_packetSize (0),
```

```
1141      m_nPackets (0),
1142      m_dataRate (0),
1143      m_sendEvent (),
1144      m_running (false),
1145      m_packetsSent (0),
1146      m_appcycle (1),
1147      m_acid (0),
1148      m_totalsent (0)
1149  {
1150  }
1151  TrafficApp::~TrafficApp()
1152  {
1153    m_socket = 0;
1154  }
1155  void
1156  TrafficApp::Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t nPackets, DataRate dataRate, uint32_t
1157  appcycle, uint32_t acid)
1158  {
1159    m_socket = socket;
1160    m_peer = address;
1161    m_packetSize = packetSize;
1162    m_nPackets = nPackets;
1163    m_dataRate = dataRate;
1164    m_appcycle = appcycle;
1165    m_acid = acid;
1166  }
1167  void
1168  TrafficApp::StartApplication (void)
1169  {
```

```
  m_running = true;
  m_packetsSent = 0;
  m_socket->Bind ();
  m_socket->Connect (m_peer);
  uint8_t buffer[3];
  buffer[0]= (uint8_t) DF; //data frame
  buffer[1]= (uint8_t) m_acid; //id for packet (0xff for interference)
  buffer[2]= (uint8_t) m_appcycle; //application cycle in ms for packet
  uint8_t* buf = buffer;
  m_packet = Create<Packet> (buf, m_packetSize);
  SendPacket ();
}
void
TrafficApp::StopApplication (void)
{
  m_running = false;
  NS_LOG_INFO ("TrafficApp: Total packets sent: " << m_totalsent);
  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }
  if (m_socket)
    {
      m_socket->Close ();
    }
}
void
TrafficApp::SendPacket (void)
{
```

```
  m_socket->Send (m_packet);
  m_totalsent++;
  if (++m_packetsSent < m_nPackets)
    {
      ScheduleTx ();
    }
  else
  {
        ScheduleTxCycle ();
  }
}
//Schedule sending within the communication cycle
void
TrafficApp::ScheduleTx (void)
{
  if (m_running)
    {
      //Schedule next frame
      Time tNext (Seconds (m_packetSize * 8 / static_cast<double> (m_dataRate.GetBitRate ())));
      m_sendEvent = Simulator::Schedule (tNext, &TrafficApp::SendPacket, this);
         //NS_LOG_INFO ("ScheduleTx in " << tNext);
    }
}
//Schedule sending for the application cycle
void
TrafficApp::ScheduleTxCycle (void)
{
  if (m_running)
    {
```

```
      Time tNextCycle (Seconds ((m_appcycle/static_cast<double> (1000))- m_packetsSent * (m_packetSize * 8 / static_cast<double>
(m_dataRate.GetBitRate ()))  ) );
      m_sendEvent = Simulator::Schedule (tNextCycle, &TrafficApp::SendPacket, this);
         //NS_LOG_INFO ("ScheduleTxCycle in " << tNextCycle);
    }
 m_packetsSent = 0;
}
int
main (int argc, char *argv[])
{
  //
  // explicit debugging for selected modules
  //
#if 1
  LogComponentEnable ("LDCSim", LOG_LEVEL_ALL);
#endif
#if 0
  LogComponentEnable ("CsmaNetDevice", LOG_LEVEL_ALL);
#endif
#if 0
  LogComponentEnable ("OnOffApplication", LOG_LEVEL_ALL);
#endif
#if 0
  LogComponentEnable ("Ipv4EndPoint", LOG_LEVEL_ALL);
#endif
#if 0
  LogComponentEnable ("UdpSocketImpl", LOG_LEVEL_ALL);
#endif
#if 0
```

```
  LogComponentEnable ("Simulator", LOG_LEVEL_ALL);
#endif
#if 0
  LogComponentEnable ("Application", LOG_LEVEL_ALL);
#endif
#if 0
  LogComponentEnable ("OnOffApplication", LOG_LEVEL_ALL);
#endif
  //
  //command-line arguments
  //
  CommandLine cmd (__FILE__);
  cmd.Parse (argc, argv);
  //
  // Explicitly create the nodes required by the topology (shown above).
  //
  NS_LOG_INFO ("rolling mean window size = " << RM_WINDOWSIZE);
  NS_LOG_INFO ("Create nodes.");
  Ptr<Node> AC1 = CreateObject<Node> ();
  Ptr<Node> n1 = CreateObject<Node> ();
  Ptr<Node> n2 = CreateObject<Node> ();
  Ptr<Node> n3 = CreateObject<Node> ();
  Ptr<Node> n4 = CreateObject<Node> ();
  Ptr<Node> n5 = CreateObject<Node> ();
  Ptr<Node> n6 = CreateObject<Node> ();
  Ptr<Node> n7 = CreateObject<Node> ();
  Ptr<Node> n8 = CreateObject<Node> ();
  Ptr<Node> n9 = CreateObject<Node> ();
  NS_LOG_INFO ("Build Topology");
```

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("1000Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (0.002)));
// Create the NetDevice containers for the csma links
NetDeviceContainer Lan1Devices;
NetDeviceContainer Lan2Devices;
NetDeviceContainer Lan3Devices;
NetDeviceContainer Lan4Devices;
NetDeviceContainer Lan5Devices;
NetDeviceContainer Lan6Devices;
NetDeviceContainer Lan7Devices;
NetDeviceContainer Lan8Devices;
NetDeviceContainer Lan9Devices;
NetDeviceContainer Lan10Devices;
// put the nodes into containers for the LAN segments
NodeContainer Lan1 (AC1, n1);
NodeContainer Lan2 (n1, n2);
NodeContainer Lan3 (n2, n3);
NodeContainer Lan4 (n3, n4);
NodeContainer Lan5 (n4, n5);
NodeContainer Lan6 (n5, n6);
NodeContainer Lan7 (n6, n7);
NodeContainer Lan8 (n7, n8);
NodeContainer Lan9 (n8, n9);
NodeContainer Lan10 (n9, AC1);
NodeContainer Seg1 (AC1, n1, n2, n3, n4);
NodeContainer Seg2 (n5, n6, n7, n8, n9);
NodeContainer AllSeg (Seg1, Seg2);
// positions for the use of NetAnim
```

```
  AnimationInterface::SetConstantPosition (Lan1.Get (0), 10, 50);
  AnimationInterface::SetConstantPosition (n1, 20, 30);
  AnimationInterface::SetConstantPosition (n2, 30, 30);
  AnimationInterface::SetConstantPosition (n3, 40, 30);
  AnimationInterface::SetConstantPosition (n4, 50, 30);
  AnimationInterface::SetConstantPosition (n5, 60, 50);
  AnimationInterface::SetConstantPosition (n6, 50, 70);
  AnimationInterface::SetConstantPosition (n7, 40, 70);
  AnimationInterface::SetConstantPosition (n8, 30, 70);
  AnimationInterface::SetConstantPosition (n9, 20, 70);
//Create the csma links
  Lan1Devices = csma.Install (NodeContainer (Lan1.Get (0), Lan1.Get (1)));
  Lan2Devices = csma.Install (NodeContainer (Lan2.Get (0), Lan2.Get (1)));
  Lan3Devices = csma.Install (NodeContainer (Lan3.Get (0), Lan3.Get (1)));
  Lan4Devices = csma.Install (NodeContainer (Lan4.Get (0), Lan4.Get (1)));
  Lan5Devices = csma.Install (NodeContainer (Lan5.Get (0), Lan5.Get (1)));
  Lan6Devices = csma.Install (NodeContainer (Lan6.Get (0), Lan6.Get (1)));
  Lan7Devices = csma.Install (NodeContainer (Lan7.Get (0), Lan7.Get (1)));
  Lan8Devices = csma.Install (NodeContainer (Lan8.Get (0), Lan8.Get (1)));
  Lan9Devices = csma.Install (NodeContainer (Lan9.Get (0), Lan9.Get (1)));
  Lan10Devices = csma.Install (NodeContainer (Lan10.Get (0), Lan10.Get (1)));
  //build new device containers for the bridge devices
    NetDeviceContainer Bridge_n1Devices;
    NetDeviceContainer Bridge_n2Devices;
    NetDeviceContainer Bridge_n3Devices;
    NetDeviceContainer Bridge_n4Devices;
    NetDeviceContainer Bridge_n5Devices;
    NetDeviceContainer Bridge_n6Devices;
    NetDeviceContainer Bridge_n7Devices;
```

```
NetDeviceContainer Bridge_n8Devices;
NetDeviceContainer Bridge_n9Devices;
NetDeviceContainer AllDevices;
Bridge_n1Devices.Add (Lan1Devices.Get(1));
Bridge_n1Devices.Add (Lan2Devices.Get(0));
Bridge_n2Devices.Add (Lan2Devices.Get(1));
Bridge_n2Devices.Add (Lan3Devices.Get(0));
Bridge_n3Devices.Add (Lan3Devices.Get(1));
Bridge_n3Devices.Add (Lan4Devices.Get(0));
Bridge_n4Devices.Add (Lan4Devices.Get(1));
Bridge_n4Devices.Add (Lan5Devices.Get(0));
Bridge_n5Devices.Add (Lan5Devices.Get(1));
Bridge_n5Devices.Add (Lan6Devices.Get(0));
Bridge_n6Devices.Add (Lan6Devices.Get(1));
Bridge_n6Devices.Add (Lan7Devices.Get(0));
Bridge_n7Devices.Add (Lan7Devices.Get(1));
Bridge_n7Devices.Add (Lan8Devices.Get(0));
Bridge_n8Devices.Add (Lan8Devices.Get(1));
Bridge_n8Devices.Add (Lan9Devices.Get(0));
Bridge_n9Devices.Add (Lan9Devices.Get(1));
Bridge_n9Devices.Add (Lan10Devices.Get(0));
AllDevices.Add (Lan1Devices.Get(0));
AllDevices.Add (Lan1Devices.Get(1));
AllDevices.Add (Lan2Devices.Get(0));
AllDevices.Add (Lan2Devices.Get(1));
AllDevices.Add (Lan3Devices.Get(0));
AllDevices.Add (Lan3Devices.Get(1));
AllDevices.Add (Lan4Devices.Get(0));
AllDevices.Add (Lan4Devices.Get(1));
```

```
  AllDevices.Add (Lan5Devices.Get(0));
  AllDevices.Add (Lan5Devices.Get(1));
  AllDevices.Add (Lan6Devices.Get(0));
  AllDevices.Add (Lan6Devices.Get(1));
  AllDevices.Add (Lan7Devices.Get(0));
  AllDevices.Add (Lan7Devices.Get(1));
  AllDevices.Add (Lan8Devices.Get(0));
  AllDevices.Add (Lan8Devices.Get(1));
  AllDevices.Add (Lan9Devices.Get(0));
  AllDevices.Add (Lan9Devices.Get(1));
  AllDevices.Add (Lan10Devices.Get(0));
  AllDevices.Add (Lan10Devices.Get(1));
//
// Create the bridge NetDevice for packet switching.
// Each node contains a bridge thereby forming the ring with two-port devices.
//
BridgeHelper Bridge_n1, Bridge_n2, Bridge_n3, Bridge_n4, Bridge_n5;
BridgeHelper Bridge_n6, Bridge_n7, Bridge_n8, Bridge_n9;
Bridge_n1.Install (n1, Bridge_n1Devices);
Bridge_n2.Install (n2, Bridge_n2Devices);
Bridge_n3.Install (n3, Bridge_n3Devices);
Bridge_n4.Install (n4, Bridge_n4Devices);
Bridge_n5.Install (n5, Bridge_n5Devices);
Bridge_n6.Install (n6, Bridge_n6Devices);
Bridge_n7.Install (n7, Bridge_n7Devices);
Bridge_n8.Install (n8, Bridge_n8Devices);
Bridge_n9.Install (n9, Bridge_n9Devices);
// Add internet stack to the nodes. Stack is per node, not per NetDevice (interface)
InternetStackHelper internet;
```

```
  internet.Install (AllSeg);
  // "hardware" in place.  Add IP addresses.
  // IP address is per NetDevice (interface) starting from PLC with 10.1.1.1
  // n1 clockwise (cw): 10.1.1.2               n6 cw:  10.1.1.12
  // n1 counterclockwise (ccw): 10.1.1.3       n6 ccw: 10.1.1.13
  // n2 cw: 10.1.1.4                           n7 cw:  10.1.1.14
  // n2 ccw: 10.1.1.5                          n7 ccw: 10.1.1.15
  // n3 cw: 10.1.1.6                           n8 cw:  10.1.1.16
  // n3 ccw: 10.1.1.7                          n8 ccw: 10.1.1.17
  // n4 cw: 10.1.1.8                           n9 cw:  10.1.1.18
  // n4 ccw: 10.1.1.9                          n9 ccw: 10.1.1.19
  // n5 cw: 10.1.1.10                          PLC cw: 10.1.1.20
  // n5 ccw: 10.1.1.11                         PLC ccw:10.1.1.1
  //  etc....
  NS_LOG_INFO ("Assign IP Addresses.");
  Ipv4AddressHelper ipv4;
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  ipv4.Assign (AllDevices);
  //
  // Create router nodes, initialize routing database and set up the routing
  // tables in the nodes.  We excuse the bridge nodes from having to serve as
  // routers, since they don't even have internet stacks on them.
  //
  //Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
   NS_LOG_INFO ("Create Applications.");
   uint16_t port = 9;   // Discard port (RFC 863)
   //*************Create AC 1 Applications************
   //************* 1 ms clockwise round the ring************
#if (1)
```

```
   Ptr<Socket> LDCcw1msSocket = Socket::CreateSocket (Lan1.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCcw1msApp = CreateObject<LDCApp> ();
   Lan1.Get (0)->AddApplication (LDCcw1msApp);
   LDCcw1msSocket->BindToNetDevice (Lan1Devices.Get(0));
   LDCcw1msApp->SetStartTime (Seconds (SIMSTART));
   LDCcw1msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
   //   Add the throughput collection and maximum calculation app.
   Ptr<CollApp> CollAC1App = CreateObject<CollApp> ();
   Lan1.Get (0)->AddApplication (CollAC1App);
   Lan1Devices.Get (0)->TraceConnectWithoutContext ("MacRx", MakeCallback (&CollApp::ReceivePacket, CollAC1App));
   CollAC1App->Setup(N0);
   CollAC1App->SetStartTime (Seconds (SIMSTART + COLL_STARTDELAY));
   CollAC1App->SetStopTime (Seconds (SIMEND + COLL_ENDDELAY));
#ifndef NOCONTROL
#ifdef COMMONCONTROL
   //install and setup distribution controller for all app cycles
   Ptr<PIDCtrlApp> DistCtrlAC1_AllApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (DistCtrlAC1_AllApp);
   DistCtrlAC1_AllApp->Setup (0.2, 40, 0/*0.00001*/, 1000000, 0.0, &LDCcw1msApp->m_distctrlin /*&CollAC1App-
>m_thp_array[APP1MS][CW][NNODES+1]*/ , &LDCcw1msApp->m_distctrlout, "Distribution");
   DistCtrlAC1_AllApp->SetStartTime (Seconds (CTRL_SIMSTART+ DIST_STARTDELAY));
   DistCtrlAC1_AllApp->SetStopTime (Seconds (SIMEND + DIST_ENDDELAY));
   //install and setup flow controller
   Ptr<PIDCtrlApp> FlowCtrlAC1_AllApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (FlowCtrlAC1_AllApp);
   FlowCtrlAC1_AllApp->Setup (0.6, 48/*85*/, 0.000/*0.00029*/, 1000000, 0.0, &LDCcw1msApp->m_flowctrlin , &LDCcw1msApp-
>m_flowctrlout, "Flow");
   FlowCtrlAC1_AllApp->SetStartTime (Seconds (CTRL_SIMSTART+ FLOW_STARTDELAY));
   FlowCtrlAC1_AllApp->SetStopTime (Seconds (SIMEND+ FLOW_ENDDELAY));
```

```
   LDCcw1msApp->Setup (LDCcw1msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.18"), port)), 222, 40, DataRate ("1Gb/s"),
APP1MS, 1, false, 0, &CollAC1App->m_thp_array[ALLAPPSIND][CW][NNODES+1]);
#endif
#ifndef COMMONCONTROL
   //install and setup distribution controller
   Ptr<PIDCtrlApp> DistCtrlAC1_1msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (DistCtrlAC1_1msApp);
   DistCtrlAC1_1msApp->Setup (0.2, 40, 0.00001, 1000000, 0.01, &LDCcw1msApp->m_distctrlin /*&CollAC1App-
>m_thp_array[APP1MS][CW][NNODES+1]*/ , &LDCcw1msApp->m_distctrlout, "Distribution");
   DistCtrlAC1_1msApp->SetStartTime (Seconds (CTRL_SIMSTART+ DIST_STARTDELAY));
   DistCtrlAC1_1msApp->SetStopTime (Seconds (SIMEND + DIST_ENDDELAY));
   //install and setup flow controller
   Ptr<PIDCtrlApp> FlowCtrlAC1_1msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (FlowCtrlAC1_1msApp);
   FlowCtrlAC1_1msApp->Setup (0.4,140, 0.00002, 1000000, 0.0, &LDCcw1msApp->m_flowctrlin , &LDCcw1msApp->m_flowctrlout, "Flow");
   FlowCtrlAC1_1msApp->SetStartTime (Seconds (CTRL_SIMSTART+ FLOW_STARTDELAY));
   FlowCtrlAC1_1msApp->SetStopTime (Seconds (SIMEND+ FLOW_ENDDELAY));
   LDCcw1msApp->Setup (LDCcw1msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.18"), port)), 222, 15, DataRate ("1Gb/s"),
APP1MS, 1, false, 0, &CollAC1App->m_thp_array[APP1MSIND][CW][NNODES+1]);
#endif
#endif
   NS_LOG_INFO ("Application 1 ms clockwise created.");
#endif
   //************* 2 ms clockwise round the ring************
#if (1)
   Ptr<Socket> LDCcw2msSocket = Socket::CreateSocket (Lan1.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCcw2msApp = CreateObject<LDCApp> ();
   Lan1.Get (0)->AddApplication (LDCcw2msApp);
   LDCcw2msSocket->BindToNetDevice (Lan1Devices.Get(0));
```

```
   LDCcw2msApp->SetStartTime (Seconds (SIMSTART));
   LDCcw2msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
#if 0
   //install and setup distribution controller
   Ptr<PIDCtrlApp> DistCtrlAC1_1msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (DistCtrlAC1_1msApp);
   DistCtrlAC1_1msApp->Setup (0.2, 40, 0.00001, 1000000, 0.01, &LDCcw1msApp->m_distctrlin /*&CollAC1App-
>m_thp_array[APP1MS][CW][NNODES+1]*/ , &LDCcw1msApp->m_distctrlout, "Distribution");
   DistCtrlAC1_1msApp->SetStartTime (Seconds (CTRL_SIMSTART+ DIST_STARTDELAY));
   DistCtrlAC1_1msApp->SetStopTime (Seconds (SIMEND + DIST_ENDDELAY));
#endif
   //install and setup flow controller
   Ptr<PIDCtrlApp> FlowCtrlAC1_2msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (FlowCtrlAC1_2msApp);
   FlowCtrlAC1_2msApp->Setup (0.4, 145, 0.00002, 1000000, 0.0, &LDCcw2msApp->m_flowctrlin , &LDCcw2msApp->m_flowctrlout, "Flow");
   FlowCtrlAC1_2msApp->SetStartTime (Seconds (CTRL_SIMSTART+ FLOW_STARTDELAY));
   FlowCtrlAC1_2msApp->SetStopTime (Seconds (SIMEND+ FLOW_ENDDELAY));
   LDCcw2msApp->Setup (LDCcw2msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.18"), port)), 223, 15, DataRate ("1Gb/s"),
APP2MS, 1, false, 0, &CollAC1App->m_thp_array[APP2MSIND][CW][NNODES+1]);
   NS_LOG_INFO ("Application 2 ms clockwise created.");
#endif
   //************* 4 ms clockwise round the ring************
#if (1)
   Ptr<Socket> LDCcw4msSocket = Socket::CreateSocket (Lan1.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCcw4msApp = CreateObject<LDCApp> ();
   Lan1.Get (0)->AddApplication (LDCcw4msApp);
   LDCcw4msSocket->BindToNetDevice (Lan1Devices.Get(0));
   LDCcw4msApp->SetStartTime (Seconds (SIMSTART));
   LDCcw4msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
```

```
#if 0
   //install and setup distribution controller
   Ptr<PIDCtrlApp> DistCtrlAC1_1msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (DistCtrlAC1_1msApp);
   DistCtrlAC1_1msApp->Setup (0.2, 40, 0.00001, 1000000, 0.01, &LDCcw1msApp->m_distctrlin /*&CollAC1App-
>m_thp_array[APP1MS][CW][NNODES+1]*/ , &LDCcw1msApp->m_distctrlout, "Distribution");
   DistCtrlAC1_1msApp->SetStartTime (Seconds (CTRL_SIMSTART+ DIST_STARTDELAY));
   DistCtrlAC1_1msApp->SetStopTime (Seconds (SIMEND + DIST_ENDDELAY));
#endif
   //install and setup flow controller
   Ptr<PIDCtrlApp> FlowCtrlAC1_4msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (FlowCtrlAC1_4msApp);
   FlowCtrlAC1_4msApp->Setup (0.4, 160, 0.00002, 1000000, 0.0, &LDCcw4msApp->m_flowctrlin , &LDCcw4msApp->m_flowctrlout, "Flow");
   FlowCtrlAC1_4msApp->SetStartTime (Seconds (CTRL_SIMSTART+ FLOW_STARTDELAY));
   FlowCtrlAC1_4msApp->SetStopTime (Seconds (SIMEND+ FLOW_ENDDELAY));
   LDCcw4msApp->Setup (LDCcw4msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.18"), port)), 224, 20, DataRate ("1Gb/s"),
APP4MS, 1, false, 0, &CollAC1App->m_thp_array[APP4MSIND][CW][NNODES+1]);
   NS_LOG_INFO ("Application 4 ms clockwise created.");
#endif
   //************* 8 ms clockwise round the ring************
#if (1)
   Ptr<Socket> LDCcw8msSocket = Socket::CreateSocket (Lan1.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCcw8msApp = CreateObject<LDCApp> ();
   Lan1.Get (0)->AddApplication (LDCcw8msApp);
   LDCcw8msSocket->BindToNetDevice (Lan1Devices.Get(0));
   LDCcw8msApp->SetStartTime (Seconds (SIMSTART));
   LDCcw8msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
#if 0
   //install and setup distribution controller
```

```
   Ptr<PIDCtrlApp> DistCtrlAC1_1msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (DistCtrlAC1_1msApp);
   DistCtrlAC1_1msApp->Setup (0.2, 40, 0.00001, 1000000, 0.01, &LDCcw1msApp->m_distctrlin /*&CollAC1App-
>m_thp_array[APP1MS][CW][NNODES+1]*/ , &LDCcw1msApp->m_distctrlout, "Distribution");
   DistCtrlAC1_1msApp->SetStartTime (Seconds (CTRL_SIMSTART+ DIST_STARTDELAY));
   DistCtrlAC1_1msApp->SetStopTime (Seconds (SIMEND + DIST_ENDDELAY));
#endif
   //install and setup flow controller
   Ptr<PIDCtrlApp> FlowCtrlAC1_8msApp = CreateObject<PIDCtrlApp> ();
   Lan1.Get (0)->AddApplication (FlowCtrlAC1_8msApp);
   FlowCtrlAC1_8msApp->Setup (0.4, 220, 0.00002, 1000000, 0.0, &LDCcw8msApp->m_flowctrlin , &LDCcw8msApp->m_flowctrlout, "Flow");
   FlowCtrlAC1_8msApp->SetStartTime (Seconds (CTRL_SIMSTART+ FLOW_STARTDELAY));
   FlowCtrlAC1_8msApp->SetStopTime (Seconds (SIMEND+ FLOW_ENDDELAY));
   LDCcw8msApp->Setup (LDCcw8msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.18"), port)), 225, 25, DataRate ("1Gb/s"),
APP8MS, 1, false, 0, &CollAC1App->m_thp_array[APP8MSIND][CW][NNODES+1]);
   NS_LOG_INFO ("Application 8 ms clockwise created.");
#endif
   //************* 1 ms counterclockwise round the ring************
   Ptr<Socket> LDCccw1msSocket = Socket::CreateSocket (Lan10.Get (1), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCccw1msApp = CreateObject<LDCApp> ();
   LDCccw1msApp->Setup (LDCccw1msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.3"), port)), 211, 15, DataRate ("1Gb/s"),
APP1MS, 1, true, &LDCcw1msApp->m_flowctrlout, &CollAC1App->m_thp_array[APP1MS][CCW][NNODES+1]);
   Lan1.Get (0)->AddApplication (LDCccw1msApp);
   LDCccw1msSocket->BindToNetDevice (Lan10Devices.Get(1));
   LDCccw1msApp->SetStartTime (Seconds (SIMSTART));
   LDCccw1msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
   Lan10Devices.Get (1)->TraceConnectWithoutContext ("MacRx", MakeCallback (&CollApp::ReceivePacket, CollAC1App));
   NS_LOG_INFO ("Application 1 ms counterclockwise created.");
   //************* 2 ms counterclockwise round the ring************
```

```
#if 1
   Ptr<Socket> LDCccw2msSocket = Socket::CreateSocket (Lan10.Get (1), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCccw2msApp = CreateObject<LDCApp> ();
   LDCccw2msApp->Setup (LDCccw2msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.3"), port)), 212, 15, DataRate ("1Gb/s"),
APP2MS, 1, true, &LDCcw2msApp->m_flowctrlout, &CollAC1App->m_thp_array[APP2MS][CCW][NNODES+1]);
   Lan1.Get (0)->AddApplication (LDCccw2msApp);
   LDCccw2msSocket->BindToNetDevice (Lan10Devices.Get(1));
   LDCccw2msApp->SetStartTime (Seconds (SIMSTART));
   LDCccw2msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
   Lan10Devices.Get (1)->TraceConnectWithoutContext ("MacRx", MakeCallback (&CollApp::ReceivePacket, CollAC1App));
   NS_LOG_INFO ("Application 2 ms counterclockwise created.");
   //************* 4 ms counterclockwise round the ring************
   Ptr<Socket> LDCccw4msSocket = Socket::CreateSocket (Lan10.Get (1), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCccw4msApp = CreateObject<LDCApp> ();
   LDCccw4msApp->Setup (LDCccw4msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.3"), port)), 213, 20, DataRate ("1Gb/s"),
APP4MS, 1, true, &LDCcw4msApp->m_flowctrlout, &CollAC1App->m_thp_array[APP4MS][CCW][NNODES+1]);
   Lan1.Get (0)->AddApplication (LDCccw4msApp);
   LDCccw4msSocket->BindToNetDevice (Lan10Devices.Get(1));
   LDCccw4msApp->SetStartTime (Seconds (SIMSTART));
   LDCccw4msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
   Lan10Devices.Get (1)->TraceConnectWithoutContext ("MacRx", MakeCallback (&CollApp::ReceivePacket, CollAC1App));
   NS_LOG_INFO ("Application 4 ms counterclockwise created.");
   //************* 8 ms counterclockwise round the ring************
   Ptr<Socket> LDCccw8msSocket = Socket::CreateSocket (Lan10.Get (1), UdpSocketFactory::GetTypeId ());
   Ptr<LDCApp> LDCccw8msApp = CreateObject<LDCApp> ();
   LDCccw8msApp->Setup (LDCccw8msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.3"), port)), 214, 25, DataRate ("1Gb/s"),
APP8MS, 1, true, &LDCcw8msApp->m_flowctrlout, &CollAC1App->m_thp_array[APP8MS][CCW][NNODES+1]);
   Lan1.Get (0)->AddApplication (LDCccw8msApp);
   LDCccw8msSocket->BindToNetDevice (Lan10Devices.Get(1));
```

```
   LDCccw8msApp->SetStartTime (Seconds (SIMSTART));
   LDCccw8msApp->SetStopTime (Seconds (SIMEND + LDC_ENDDELAY));
   Lan10Devices.Get (1)->TraceConnectWithoutContext ("MacRx", MakeCallback (&CollApp::ReceivePacket, CollAC1App));
   NS_LOG_INFO ("Application 8 ms counterclockwise created.");
#endif
   //*************Create Interference Loads************
   //************* Interference 1 on n2 clockwise to n4, cycle 1 ms ************
#ifdef INTERFERENCE_1MS
   Ptr<Socket> IFn2n4cw1msSocket = Socket::CreateSocket (Lan2.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<TrafficApp> IFn2n4cw1msApp = CreateObject<TrafficApp> ();
   IFn2n4cw1msApp->Setup (IFn2n4cw1msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.8"), port)), 200, 15, DataRate
("1Gb/s"), APP1MS, 1);
   Lan2.Get (0)->AddApplication (IFn2n4cw1msApp);
   IFn2n4cw1msSocket->BindToNetDevice (Lan2Devices.Get(0));
   IFn2n4cw1msApp->SetStartTime (Seconds (INTERFERENCE_START_1MS));
   IFn2n4cw1msApp->SetStopTime (Seconds (SIMEND/*INTERFERENCE_STOP_1MS*/));
   NS_LOG_INFO ("Interference clockwise from n2 to n4 created.");
#endif
   //************* Interference 2 on n2 clockwise to n4, cycle 2 ms ************
#ifdef INTERFERENCE_2MS
   Ptr<Socket> IFn2n4cw2msSocket = Socket::CreateSocket (Lan2.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<TrafficApp> IFn2n4cw2msApp = CreateObject<TrafficApp> ();
   IFn2n4cw2msApp->Setup (IFn2n4cw2msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.8"), port)), 200, 20, DataRate
("1Gb/s"), APP2MS, 1);
   Lan2.Get (0)->AddApplication (IFn2n4cw2msApp);
   IFn2n4cw2msSocket->BindToNetDevice (Lan2Devices.Get(0));
   IFn2n4cw2msApp->SetStartTime (Seconds (INTERFERENCE_START_2MS));
   IFn2n4cw2msApp->SetStopTime (Seconds (SIMEND));
   NS_LOG_INFO ("Interference clockwise from n2 to n4 created.");
```

```
#endif
   //************* Interference 3 on n1 clockwise to n5, cycle 4 ms ************
#ifdef INTERFERENCE_4MS
   Ptr<Socket> IFn1n5cw4msSocket = Socket::CreateSocket (Lan2.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<TrafficApp> IFn1n5cw4msApp = CreateObject<TrafficApp> ();
   IFn1n5cw4msApp->Setup (IFn1n5cw4msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.10"), port)), 200, 30, DataRate
("1Gb/s"), APP4MS, 1);
   Lan2.Get (0)->AddApplication (IFn1n5cw4msApp);
   IFn1n5cw4msSocket->BindToNetDevice (Lan2Devices.Get(0));
   IFn1n5cw4msApp->SetStartTime (Seconds (INTERFERENCE_START_4MS));
   IFn1n5cw4msApp->SetStopTime (Seconds (SIMEND));
   NS_LOG_INFO ("Interference clockwise from n1 to n5 created.");
#endif
   //************* Interference 4 on n1 clockwise to n5, cycle 8 ms ************
#ifdef INTERFERENCE_8MS
   Ptr<Socket> IFn1n5cw8msSocket = Socket::CreateSocket (Lan2.Get (0), UdpSocketFactory::GetTypeId ());
   Ptr<TrafficApp> IFn1n5cw8msApp = CreateObject<TrafficApp> ();
   IFn1n5cw8msApp->Setup (IFn1n5cw8msSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.10"), port)), 200, 40, DataRate
("1Gb/s"), APP8MS, 1);
   Lan2.Get (0)->AddApplication (IFn1n5cw8msApp);
   IFn1n5cw8msSocket->BindToNetDevice (Lan2Devices.Get(0));
   IFn1n5cw8msApp->SetStartTime (Seconds (INTERFERENCE_START_8MS));
   IFn1n5cw8msApp->SetStopTime (Seconds (SIMEND));
   NS_LOG_INFO ("Interference clockwise from n1 to n5 created.");
#endif
   //*************Create rolling mean throughput measurement applications************
   //*** on n3 clockwise***
#if 1
   // 1 ms integration time
```

```
   Ptr<RollMeanApp> RollMeanAC1APP1I1_n3d1App = CreateObject<RollMeanApp> ();
   Lan4.Get (0)->AddApplication (RollMeanAC1APP1I1_n3d1App);
   Ptr<Socket> RollMeanAC1APP1I1_n3d1AppSocket = Socket::CreateSocket (Lan4.Get (0), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP1I1_n3d1App->Setup (RollMeanAC1APP1I1_n3d1AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.20"), port)),
60, DataRate ("1Gb/s"), CW, APP1MS, INT1MS, N3, true);
   RollMeanAC1APP1I1_n3d1AppSocket->BindToNetDevice (Lan4Devices.Get(0));
   RollMeanAC1APP1I1_n3d1App->SetStartTime (Seconds (SIMSTART + RM_1MS_DELAY));
   RollMeanAC1APP1I1_n3d1App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean 1 ms on node 3 clockwise created.");
   Lan4Devices.Get (0)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP1I1_n3d1App));
#if 1
   // 2 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP2I2_n3d1App = CreateObject<RollMeanApp> ();
   Lan4.Get (0)->AddApplication (RollMeanAC1APP2I2_n3d1App);
   Ptr<Socket> RollMeanAC1APP2I2_n3d1AppSocket = Socket::CreateSocket (Lan4.Get (0), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP2I2_n3d1App->Setup (RollMeanAC1APP2I2_n3d1AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.20"), port)),
60, DataRate ("1Gb/s"), CW, APP2MS, INT2MS, N3, true);
   RollMeanAC1APP2I2_n3d1AppSocket->BindToNetDevice (Lan4Devices.Get(0));
   RollMeanAC1APP2I2_n3d1App->SetStartTime (Seconds (SIMSTART + RM_2MS_DELAY));
   RollMeanAC1APP2I2_n3d1App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean 2 ms on node 3 clockwise created.");
   Lan4Devices.Get (0)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP2I2_n3d1App));
   // 4 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP4I4_n3d1App = CreateObject<RollMeanApp> ();
   Lan4.Get (0)->AddApplication (RollMeanAC1APP4I4_n3d1App);
   Ptr<Socket> RollMeanAC1APP4I4_n3d1AppSocket = Socket::CreateSocket (Lan4.Get (0), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP4I4_n3d1App->Setup (RollMeanAC1APP4I4_n3d1AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.20"), port)),
60, DataRate ("1Gb/s"), CW, APP4MS, INT4MS, N3, true);
   RollMeanAC1APP4I4_n3d1AppSocket->BindToNetDevice (Lan4Devices.Get(0));
```

```
   RollMeanAC1APP4I4_n3d1App->SetStartTime (Seconds (SIMSTART + RM_4MS_DELAY));
   RollMeanAC1APP4I4_n3d1App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean 4 ms on node 3 clockwise created.");
   Lan4Devices.Get (0)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP4I4_n3d1App));
   // 8 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP8I8_n3d1App = CreateObject<RollMeanApp> ();
   Lan4.Get (0)->AddApplication (RollMeanAC1APP8I8_n3d1App);
   Ptr<Socket> RollMeanAC1APP8I8_n3d1AppSocket = Socket::CreateSocket (Lan4.Get (0), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP8I8_n3d1App->Setup (RollMeanAC1APP8I8_n3d1AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.20"), port)),
60, DataRate ("1Gb/s"), CW, APP8MS, INT8MS, N3, true);
   RollMeanAC1APP8I8_n3d1AppSocket->BindToNetDevice (Lan4Devices.Get(0));
   RollMeanAC1APP8I8_n3d1App->SetStartTime (Seconds (SIMSTART + RM_8MS_DELAY));
   RollMeanAC1APP8I8_n3d1App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean 8 ms on node 3 clockwise created.");
   Lan4Devices.Get (0)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP8I8_n3d1App));
#endif
#endif
#if 0
     // 32 ms integration time
      Ptr<RollMeanApp> RollMeanAC1APP32I32_n3d1App = CreateObject<RollMeanApp> ();
      Lan4.Get (0)->AddApplication (RollMeanAC1APP32I32_n3d1App);
      Ptr<Socket> RollMeanAC1APP32I32_n3d1AppSocket = Socket::CreateSocket (Lan4.Get (0), UdpSocketFactory::GetTypeId ());
      RollMeanAC1APP32I32_n3d1App->Setup (RollMeanAC1APP32I32_n3d1AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.20"),
port)), 60, DataRate ("1Gb/s"), CW, APP1MS, 32/*INT8MS*/, N3, true);
      RollMeanAC1APP32I32_n3d1AppSocket->BindToNetDevice (Lan4Devices.Get(0));
      RollMeanAC1APP32I32_n3d1App->SetStartTime (Seconds (SIMSTART /* + RM_1MS_DELAY*/));
      RollMeanAC1APP32I32_n3d1App->SetStopTime (Seconds (SIMEND /*+ RM_ENDDELAY*/));
      NS_LOG_INFO ("Rolling mean on node 3 clockwise created.");
```

```
      Lan4Devices.Get (0)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket,
RollMeanAC1APP32I32_n3d1App));
#endif
#if (0)
#endif
   //*** on n7 counterclockwise***
#if 1
   // 1 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP1I1_n7d0App = CreateObject<RollMeanApp> ();
   Lan7.Get (1)->AddApplication (RollMeanAC1APP1I1_n7d0App);
   Ptr<Socket> RollMeanAC1APP1I1_n7d0AppSocket = Socket::CreateSocket (Lan7.Get (1), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP1I1_n7d0App->Setup (RollMeanAC1APP1I1_n7d0AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)),
60, DataRate ("1Gb/s"), CCW, APP1MS, INT1MS, N7, true);
   RollMeanAC1APP1I1_n7d0AppSocket->BindToNetDevice (Lan7Devices.Get(1));
   RollMeanAC1APP1I1_n7d0App->SetStartTime (Seconds (SIMSTART + RM_1MS_DELAY));
   RollMeanAC1APP1I1_n7d0App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean on node 7 counterclockwise created.");
   Lan7Devices.Get (1)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP1I1_n7d0App));
#endif
#if 1
   // 2 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP2I2_n7d0App = CreateObject<RollMeanApp> ();
   Lan7.Get (1)->AddApplication (RollMeanAC1APP2I2_n7d0App);
   Ptr<Socket> RollMeanAC1APP2I2_n7d0AppSocket = Socket::CreateSocket (Lan7.Get (1), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP2I2_n7d0App->Setup (RollMeanAC1APP2I2_n7d0AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)),
60, DataRate ("1Gb/s"), CCW, APP2MS, INT2MS, N7, true);
   RollMeanAC1APP2I2_n7d0AppSocket->BindToNetDevice (Lan7Devices.Get(1));
   RollMeanAC1APP2I2_n7d0App->SetStartTime (Seconds (SIMSTART + RM_2MS_DELAY));
   RollMeanAC1APP2I2_n7d0App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
```

```
   NS_LOG_INFO ("Rolling mean on node 7 counterclockwise created.");
   Lan7Devices.Get (1)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP2I2_n7d0App));
   // 4 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP4I4_n7d0App = CreateObject<RollMeanApp> ();
   Lan7.Get (1)->AddApplication (RollMeanAC1APP4I4_n7d0App);
   Ptr<Socket> RollMeanAC1APP4I4_n7d0AppSocket = Socket::CreateSocket (Lan7.Get (1), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP4I4_n7d0App->Setup (RollMeanAC1APP4I4_n7d0AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)),
60, DataRate ("1Gb/s"), CCW, APP4MS, INT4MS, N7, true);
   RollMeanAC1APP4I4_n7d0AppSocket->BindToNetDevice (Lan7Devices.Get(1));
   RollMeanAC1APP4I4_n7d0App->SetStartTime (Seconds (SIMSTART + RM_4MS_DELAY));
   RollMeanAC1APP4I4_n7d0App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean on node 7 counterclockwise created.");
   Lan7Devices.Get (1)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP4I4_n7d0App));
   // 8 ms integration time
   Ptr<RollMeanApp> RollMeanAC1APP8I8_n7d0App = CreateObject<RollMeanApp> ();
   Lan7.Get (1)->AddApplication (RollMeanAC1APP8I8_n7d0App);
   Ptr<Socket> RollMeanAC1APP8I8_n7d0AppSocket = Socket::CreateSocket (Lan7.Get (1), UdpSocketFactory::GetTypeId ());
   RollMeanAC1APP8I8_n7d0App->Setup (RollMeanAC1APP8I8_n7d0AppSocket, Address (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)),
60, DataRate ("1Gb/s"), CCW, APP8MS, INT8MS, N7, true);
   RollMeanAC1APP8I8_n7d0AppSocket->BindToNetDevice (Lan7Devices.Get(1));
   RollMeanAC1APP8I8_n7d0App->SetStartTime (Seconds (SIMSTART + RM_8MS_DELAY));
   RollMeanAC1APP8I8_n7d0App->SetStopTime (Seconds (SIMEND + RM_ENDDELAY));
   NS_LOG_INFO ("Rolling mean on node 7 counterclockwise created.");
   Lan7Devices.Get (1)->TraceConnectWithoutContext ("MacTx", MakeCallback (&RollMeanApp::CheckInPacket, RollMeanAC1APP8I8_n7d0App));
#endif
  NS_LOG_INFO ("Configure Tracing.");
  //
  // Configure tracing of all enqueue, dequeue, and NetDevice receive events.
  // Trace output will be sent to the file .tr"
```

```
  //
  AsciiTraceHelper ascii;
  csma.EnableAsciiAll (ascii.CreateFileStream ("LDCv011.tr"));
  //
  // tcpdump traces; each interface will be traced.
  // The output files will be named:
  //     LDCv011-<nodeId>-<interfaceId>.pcap
  // and can be read by the "tcpdump -r" command (use "-tt" option to
  // display timestamps correctly)
  //
  csma.EnablePcapAll ("LDCv011", false);
#if 0
  // Set the bounding box for animation
  //csma.BoundingBox (1, 1, 100, 100);
  std::string animFile = "dlb-animation.xml" ;  // Name of file for animation output
  // Create the animation object and configure for specified output
  AnimationInterface anim (animFile);
  //Create link description for NetAnim
  anim.UpdateNodeDescription (PLC, "PLC");
  //Create link description for NetAnim
  anim.UpdateLinkDescription (PLC, n1, "Link 1");//#Todo: Check cms helper why not serving xml file with link info like ptp
  anim.EnablePacketMetadata (); // Optional
  anim.EnableIpv4L3ProtocolCounters (Seconds (0), Seconds (0.1)); // Optional
#endif
  //
  // Run Simulation.
  //
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
```

```
//  std::cout << "Animation Trace file created:" << animFile.c_str ()<< std::endl;
  //*********** Create data files for plot***************
   std::ofstream data_uc7p1_2D;
   data_uc7p1_2D.open ("data_uc7p1_2D.txt");
  // std::ofstream data_uc6p4_3D;
  // data_uc6p4_3D.open ("data_uc6p4_3D.txt");
#ifdef PLOT_2D
   // 2D plots
    // the 32 ms integration time collective measurement
   for (uint32_t j = 0; j < DATAPOINTS -4; j = j + DATASTEP)
      {
          //stop for zero values before end
//        if (CollAC1App->plot_array[SUMAPPSIND][j] != 0)
//        {
          //for result over all apps load documentation
          data_uc7p1_2D << j << " " << CollAC1App->plot_array[SUMAPPSIND][CW][j] << " " << CollAC1App-
>plot_array[SUMAPPSIND][CCW][j]<< std::endl ;
          //for single app cycle optimization
          //data_uc7p1_2D << j << " " << RollMeanAC1APP2I2_n3d1App->p_throughput[0][j] << " " << RollMeanAC1APP2I2_n7d0App-
>p_throughput[0][j]<< std::endl ;
//        }
      }
#if 0
   // the contribution to the 3D comparison plot
   for (uint32_t j = 0; j < DATAPOINTS -4; j = j + DATASTEP)
      {
          //stop for zero values before end
          if (RollMeanAC1APP1I8_n3d1App->p_throughput[APPSCOLLECTIVE][j] != 0)
          {
```

```
         data_uc6p4_3D << "4" << " " << j << " " << RollMeanAC1APP1I8_n3d1App->p_throughput[APPSCOLLECTIVE][j] << " " <<
RollMeanAC1APP1I8_n7d0App->p_throughput[APPSCOLLECTIVE][j]<< std::endl ;
         }
      }
   data_uc6p4_3D << std::endl;
#endif
#endif
#ifdef PLOT_3D
   // 3D plots
   // the 1 ms integration time collective measurement
   for (uint32_t j = 0; j < DATAPOINTS -1; j = j + DATASTEP)
      {
         //stop for zero values before end
         if (RollMeanAC1APP1I1_n3d1App->p_throughput[APPSCOLLECTIVE][j] != 0)
         {
         data_cw << "1" << " " << j << " " << RollMeanAC1APP1I1_n3d1App->p_throughput[APPSCOLLECTIVE][j] << std::endl ;
         }
      }
   data_cw << " " << std::endl ;
   // the 2 ms integration time collective measurement
    for (uint32_t j = 0; j < DATAPOINTS -1; j = j + DATASTEP)
       {
       if (RollMeanAC1APP1I2_n3d1App->p_throughput[APPSCOLLECTIVE][j] != 0)
                 {
              data_cw << "2" << " " << j << " " << RollMeanAC1APP1I2_n3d1App->p_throughput[APPSCOLLECTIVE][j] << std::endl ;
                 }
       }
    data_cw << " " << std::endl ;
    // the 4 ms integration time collective measurement
```

```
      for (uint32_t j = 0; j < DATAPOINTS -1; j = j + DATASTEP)
         {
        if (RollMeanAC1APP1I4_n3d1App->p_throughput[APPSCOLLECTIVE][j] != 0)
                {
               data_cw << "4" << " " << j << " " << RollMeanAC1APP1I4_n3d1App->p_throughput[APPSCOLLECTIVE][j] << std::endl ;
                }
         }
      data_cw << " " << std::endl ;
      // the 8 ms integration time collective measurement
        for (uint32_t j = 0; j < DATAPOINTS -1; j = j + DATASTEP)
           {
             if (RollMeanAC1APP1I8_n3d1App->p_throughput[APPSCOLLECTIVE][j] != 0)
                       {
                  data_cw << "8" << " " << j << " " << RollMeanAC1APP1I8_n3d1App->p_throughput[APPSCOLLECTIVE][j] << std::endl ;
                       }
           }
        data_cw << " " << std::endl ;
#endif
   data_uc7p1_2D.close();
//   data_uc6p4_3D.close();
   //*********** Create 2D plot file ***************
#if 1
  std::ofstream PlotFile_cw;
  PlotFile_cw.open ("PlFile_uc7p1_2D.plt");
  PlotFile_cw << "set terminal png" << std::endl;
  PlotFile_cw << "set output \"Plot_uc7p1_2D.png\"" << std::endl;
  PlotFile_cw << "set title \"Throughput, 1 AC, Mixed Appl. Interf., Dedicated Flow Control\"" << std::endl;
  PlotFile_cw << "set xlabel \"Simulation time t in ms\"" << std::endl;
  PlotFile_cw << "set ylabel \"Throughput Âµ(t) in %\"" << std::endl;
```

```
  PlotFile_cw << "set xrange [0:400]" << std::endl;
  PlotFile_cw << "set yrange [0:25]" << std::endl;
  //PlotFile_cw << "set grid" << std::endl;
#endif
#if 0  //with smoothing
  PlotFile_cw << "plot \"data_cw.txt\" using 1:2 smooth acsplines with linespoint title \"clockwise at n3\" lw 2 pi 10,\
  \"data_cw.txt\" using 1:3  smooth acsplines with linespoint title \"counterclockwise at n7\" lw 2 pi 10" << std::endl;
#endif
#if 0  //without smoothing
  PlotFile_cw << "plot \"data_cw.txt\" using 1:2 with linespoint title \"clockwise at n3\" lw 2 pi 10,\
  \"data_cw.txt\" using 1:3  with linespoint title \"counterclockwise at n7\" lw 2 pi 10" << std::endl;
#endif
#if 0
  PlotFile_cw << "plot \"data_cw.txt\" using 1:2 with linespoint title \"clockwise n3\" lw 2,\
  \"data_cw.txt\" using 1:3 with linespoint title \"counterclockwise n7\" lw 2,\
  \"data_cw.txt\" using 1:4 with linespoint title \"counterclockwise n8\" lw 2,\
  \"data_cw.txt\" using 1:5 with linespoint title \"counterclockwise n9\" lw 2" << std::endl;
#endif
#if 0
  //*********** Create 3D plot file ***************
  std::ofstream PlotFile3D_cw;
  PlotFile3D_cw.open ("PlFile_uc6p4_3D.plt");
  PlotFile3D_cw << "set terminal png font \"arial,10\"" << std::endl;
  PlotFile3D_cw << "set output \"Plot_uc6p4_3D.png\"" << std::endl;
  PlotFile3D_cw << "set title \"Throughput, 1 AC, Mixed Appl., 8 to 32 ms RM-Int., Flow Control\"" << std::endl;
  PlotFile3D_cw << "set xlabel \"\\n x: Sub-use-case Nr. 6.x\" rotate parallel" << std::endl;
  PlotFile3D_cw << "set ylabel \"\\n y: Simulation time t in ms\" rotate parallel" << std::endl;
  PlotFile3D_cw << "set zlabel \"z: Throughput Âµ(t) in %\" rotate parallel" << std::endl;
  PlotFile3D_cw << "set xrange [0:5]" << std::endl;
```

```
  PlotFile3D_cw << "set yrange [0:400]" << std::endl;
  PlotFile3D_cw << "set zrange [0:25]" << std::endl;
  PlotFile3D_cw << "set grid x y z vertical" << std::endl;
  PlotFile3D_cw << "set xyplane at 0" << std::endl;
  PlotFile3D_cw << "set view 70,55,1" << std::endl;
#endif
 #if 0
 std::ofstream PlotFile_cw;
 PlotFile_cw.open ("PlFile_cw.plt");
 PlotFile_cw << "set terminal png" << std::endl;
 PlotFile_cw << "set output \"NPackets_cw.png\"" << std::endl;
 PlotFile_cw << "set title \"Throughput, 1 AC, Mixed Appl., 1 to 8 ms RM-Int., No Control\"" << std::endl;
 PlotFile_cw << "set xlabel \"\\n x: Integration interval T_i_n_t in ms\" rotate parallel" << std::endl;
 PlotFile_cw << "set ylabel \"\\n y: Simulation time t in ms\" rotate parallel" << std::endl;
 PlotFile_cw << "set zlabel \"z: Throughput Âµ(t) in %\" rotate parallel" << std::endl;
 PlotFile_cw << "set xrange [0:8.5]" << std::endl;
 PlotFile_cw << "set yrange [0:100]" << std::endl;
 PlotFile_cw << "set zrange [0:25]" << std::endl;
 PlotFile_cw << "set grid x y z vertical" << std::endl;
 PlotFile_cw << "set xyplane at 0" << std::endl;
#endif
#if 1  //2D with smoothing
 PlotFile_cw << "plot \"data_uc7p1_2D.txt\" using 1:2 smooth acsplines with linespoint title \"clockwise at n3\" lw 2 pi 10,\
 \"data_uc7p1_2D.txt\" using 1:3  smooth acsplines with linespoint title \"counterclockwise at n7\" lw 2 pi 10" << std::endl;
#endif
#if 0  //3D without smoothing (not available in 3D)
 PlotFile3D_cw << "splot \"data_uc6p1_3D.txt\" using 1:2:3 with linespoint title \"uc6.1: No Ctrl., 8 ms RM-Int\" lw 2 pi 80,\
              \"data_uc6p2_3D.txt\" using 1:2:3 with linespoint title \"uc6.2: Flow Ctrl., 8 ms RM-Int\" lw 2 pi 80,\
              \"data_uc6p3_3D.txt\" using 1:2:3 with linespoint title \"uc6.3: Flow Ctrl., 32 ms RM-Int, no opt\" lw 2 pi 80,\
```

```
              \"data_uc6p4_3D.txt\" using 1:2:3 with linespoint title \"uc6.4: Flow Ctrl., 32 ms RM-Int, opt\" lw 2 pi 80" <<
std::endl;
#endif
#if 0  //3D without smoothing (not available in 3D
 PlotFile_cw << "splot \"data_cw.txt\" using 1:2:3 with linespoint title \"clockwise at n3\" lw 1.5 pi 10" << std::endl;
#endif
#if 0
 PlotFile_cw << "plot \"data_cw.txt\" using 1:2 with linespoint title \"clockwise n3\" lw 2,\
 \"data_cw.txt\" using 1:3 with linespoint title \"counterclockwise n7\" lw 2,\
 \"data_cw.txt\" using 1:4 with linespoint title \"counterclockwise n8\" lw 2,\
 \"data_cw.txt\" using 1:5 with linespoint title \"counterclockwise n9\" lw 2" << std::endl;
#endif
 PlotFile_cw.close();
// PlotFile3D_cw.close();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```