An opportunistic routing for automated decentralized structured complex intralogistic material flows with focus on scalability, flexibility and robustness for distribution centers in reference to an intralogistic supplier

Der Fakultät für Wirtschaftswissenschaften der
Universität Paderborn
zur Erlangung des akademischen Grades
Doktor der Wirtschaftswissenschaften
- Doctor rerum politicarum -
vorgelegte Dissertation

von

M. Sc. Lukas Kopecki
geboren am 16.12.1984 in Neustadt (Polen)

Paderborn, December 2, 2018

Dean: Prof. Dr. Caren Sureth-Sloane

Referee: Prof. Dr.-Ing. habil. Wilhelm Dangelmaier

Co-Referee: Prof. Dr. Leena Suhl

Für meine Familie

Foreword

The present work is the result of my research, which I carried out as a scholarship holder of the International Graduate School "Dynamic Intelligent Systems" of the University of Paderborn at the Chair of Business Informatics, esp. CIM of the Heinz Nixdorf Institute in cooperation with the company Lödige Industries GmbH in Warburg.

For the dedicated scientific and versatile support, I would like to thank Prof. Dr.-Ing. habil. Wilhelm Dangelmaier. He enabled me to do this work in cooperation with an industrial company, giving me the opportunity to combine research and industrial practice. I do not want to miss this valuable experience.

In addition, my thanks go to Prof. Dr. Leena Suhl for the acceptance of the second appraisal and Prof. Dr. Friedhelm Meyer auf der Heide and Prof. Dr. Stefan Betz, who gave me valuable feedback as members of my board of examiners. I thank the team of the International Graduate School "Dynamic Intelligent Systems" and their leader Prof. Dr. Eckhard Steffen for funding through a scholarship.

The interdisciplinary exchange with other scholarship holders in the numerous events of the doctoral program has greatly enriched my doctoral studies. My thanks also go to Prof. Dr.-Ing. Rudolf Lödige and the managing director of Lödige Industries GmbH, Mr. Philippe De Backer, for the opportunity for an industrial cooperation and for the opportunity to contribute and expand my knowledge in various practical projects. For the many constructive discussions and insights into the world of complex transportation systems I would like to thank especially Mr. Knud Segatz, Mr. Rainer Leichtweiss and Mr. Robert Bawn.

I would like to thank my colleagues at the chair for the motivating and working atmosphere as well as the interesting technical and non-technical discussions. It was a great time that will be remembered for a long time.

My special thanks go to my office colleagues Jens Weber and colleagues at Lödige Industries GmbH, Sebastian Lauck and Simon Boxnick. My most healthful thanks goes to my wife, who gave me the opportunity for being a father for two wonderful sons and all the patients during writing this work. I thank my sons Elias and Bennet for all the insightful walks and visits of varies places who gave me the possibility to think out of the box and rethink numerous tasks. Finally, I like to thank my parents and my three brothers as well as my parents-in-law supporting my family and me through the process of writing.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

CPU	Central Processing Unit
EPA	Enhanced Performance Architecture
ERP	Enterprise Resource Planning
HMI	Human-Machine Interface
HTTP	HyperText Transfer Protocol
I/O(s)	Input(s)/Output(s)
ISO	International Organization for Standardization
MAC	Media Access Control
MHE	Material Handling Equipment
MTR	Motor
OPC UA	Open Platform Communication Unified Architecture
OSI	Open System Interconnection
PE	Positioning Equipment
PLC	Programmable Logic Controller
QoS	Quality of Service

RM	Reconfigurable Manufacturing
SNSR	Sensor
TCP/IP	Transmission Control Protocol / Internet Protocol
TU	Transportation Unit
ULD	Unit Load Device
WMS	Warehouse Managment System

Nomenclature

$a_{\mathfrak{o}}$	Acceleration at one $\mathfrak{o} \in \mathfrak{O}$ for one container $c \in C$ that has to be moved on one $\mathfrak{m} \in \mathfrak{M}$ locally
i, j, k	Auxiliary variables
α, β, γ	Auxiliary variables representing time such that $\alpha \in \{0, ..., T\}, \beta \in \{0, ..., T\}, \gamma \in \{0, ..., T\}$
$B_{\mathfrak{m}}$	Set of jobs $\mathfrak{j} \in S_{\mathfrak{m}_i}$ at one $\mathfrak{m}_i$ with possible deadlocks with jobs in scheduler $S_{\mathfrak{m}_j}$ at $\mathfrak{m}_i$)
buffer($\mathfrak{j}_{i,\mathfrak{m}}, \mathfrak{j}_{j,\mathfrak{m}}$)	Time buffer between two jobs for two containers $i \in C, j \in C$ at one $\mathfrak{m} \in \mathfrak{M}$
c	A container that is being moved or that has to be moved in the system, such that $c \in C$
C	Set of containers
$\delta_{\mathfrak{j}}$	The completion duration of one job $\mathfrak{j} \in \mathfrak{J}$
distanceComm($\mathfrak{m}_i, \mathfrak{m}_j$)	Distance between two intelligent devices $\mathfrak{m}_i \in \mathfrak{M}$ and $\mathfrak{m}_j \in \mathfrak{M}$ on the communication level
$d_{\mathfrak{o}}$	Physical transportation distance of a container $c \in C$ that has to be moved on one $\mathfrak{m} \in \mathfrak{M}$ by an operation $\mathfrak{o}$
$\mathfrak{d}$	A possible destination for a container c, such that $\mathfrak{d} \in \mathfrak{D}$

$\mathfrak{D}$	The set of all possible destinations for a container c, with $\mathfrak{D} \subset \mathfrak{M}$
Δ	Set of samples representing the time a sequence $\mathfrak{l}$ takes for transportation of one $c \in C$ at $\mathfrak{m} \in \mathfrak{M}$ locally
durationComm$(\mathfrak{m}_i, \mathfrak{m}_j)$	Expected time to receive an answer from $\mathfrak{m}_j$ at $\mathfrak{m}_i$ for a communication request
estDuration(r_c)	Overall estimated transportation duration for c by using r_c
estCosts$(r_c, \mathfrak{m})$	Function returning the estimated costs for the route r_c needed to transport a container c from $\mathfrak{m}$ to a destination $\mathfrak{d}$ defined in r_c
e_i	An edge $e_i = (v_i, v_j) = (V_\tau \times V_\tau), v_i \in V_\tau, v_j \in V_{\tau+\alpha}, \alpha \in \{0, ..., T\}$
e_i^{-1}	An inverted edge $e_i^{-1} = (v_i, v_j)^{-1} = (v_j, v_i)$
E_τ	Set of edges $E_\tau = (V_\tau \times V_\tau)$ at time τ
$E_{\tau+\alpha}$	Set of edges from present V_τ to future configuration $E_{\tau+\alpha}$ as $E_{\tau+\alpha} = (V_\tau \times V_{\tau+\alpha})$
$f_\mathfrak{j}$	Representing a flow $E_\tau \times E_\tau$ for one job $\mathfrak{j}$ as inbound and outbound transportation for future container movement
$F_\mathfrak{m}$	Set of feasible candidates $F_\mathfrak{m}$ at $\mathfrak{m}$ for route determination
free(G_τ)	Function to check whether G_τ is deadlock-free
$\mathfrak{f}$	Transportation transition function $\mathfrak{f}(\mathfrak{l}, f_\mathfrak{j}$ connecting one local transportation sequence $\mathfrak{l}$ with a flow $f_\mathfrak{j}$ to define inbound and outbound to two neighboring PEs
$\mathfrak{F}$	Set of all transportation transition functions
$G\{0, ..., T\}$	Dynamic graph representation with snapshots $G_\tau = (V_\tau, E_\tau), \tau \in \{0, ..., T\}$

EvalWaitingTime()	Function to evaluate the waiting time for the maximum waiting time of one $\mathfrak{m}$ for a response to a seeking message s for route determination
getParent($M_c, \mathfrak{m}$)	Function to return the parent of $\mathfrak{m}$ in the communication tree M_c for backwards traversal
G_τ	A snapshot $G_\tau \in G\{0, ..., T\}$ representing one system configuration at time τ
E_u	Set of edges from the underlying graph with $E_u = i \times j, i \in V_u, j \in V_u$
G_u	Underlying graph representing the transportation network
V_u	Set of vertices of the underlying graph
hasParent($M_c, \mathfrak{m}$)	Function to check whether $\mathfrak{m}$ has a parent in the communication tree M_c for backwards traversal
$\mathfrak{h}$	Counter of hops that a message has done between two intelligent devices $\mathfrak{m}_i$ and $\mathfrak{m}_j$ being forwarded to arrive at $\mathfrak{m}_j$ has done
$H_\mathfrak{m}$	Set of blocked containers $c \in C$ at $\mathfrak{m}$ to ensure system stability by preventing deadlocks
idleTime($\mathfrak{j}_{c,\mathfrak{m}}$)	Function to determine the time $\mathfrak{j}_{c,\mathfrak{m}}$ can be delayed without violating the due date
$\mathfrak{j}_{c,\mathfrak{m}}$	Job representing an action of c at $\mathfrak{m}$ locally
$\kappa_{c,\mathfrak{m}}$	The expected completion time for one job $\mathfrak{j}_{c,\mathfrak{m}}$
LateTime($\mathfrak{j}_{c,\mathfrak{m}}$)	Lateness of one job
leaf(s)	Function returning 1 if $s \in S_c$ is a leaf, otherwise 0
l	Maximal length of r_c such that $\lvert r_c \rvert \leq l$

$\text{localSeq}(\mathfrak{m}_i, \mathfrak{m}_j)$	Returns the local sequence $\mathfrak{l}$ for moving a container c from $\mathfrak{m}_i$ to $\mathfrak{m}_j$
$\mathfrak{l}$	An ordered set $\mathfrak{l}$ is an ordered set of operations $(\mathfrak{o}_1, \ldots, \mathfrak{o}_i)$ that form a sequence of operations to move one container c from one neighbor to another neighbor of one $\mathfrak{m}$ locally
$\mathfrak{m}$	An intelligent device in the system
$\mathfrak{M}$	All intelligent devices in the system such that $\mathfrak{m} \in \mathfrak{M}$
MaxHop	System-wide maximal number of hops a message can be forwarded
MaxWaiting	System-wide maximal waiting time for a response for a sent message
$\text{minCost}(R_c)$	Determining the route r_c with the minimal costs in the temporary set R_c of collected routes for container c
$N_{\mathfrak{m}}$	Set of neighbors of $\mathfrak{m} \in \mathfrak{M}$
Ω	Ordered historical waiting times for containers C at one $\mathfrak{m} \in \mathfrak{M}$ with the order $\leq$
$\widetilde{\Omega}_{\mathfrak{m}}$	Median of the waiting times Ω for containers C at $\mathfrak{m} \in \mathfrak{M}$
$\mathfrak{O}_{\mathfrak{m}}$	Set of operations usable for movement of c by $\mathfrak{m}$
$\mathfrak{o}_{\mathfrak{m}}$	Operation on one $\mathfrak{m} \in \mathfrak{M}$ at field level, e.g. transportation, reorientation, etc...
$\text{pastCosts}(r_c)$	Returns the real costs to move the container c based on the route r_c
$\psi_{c,\mathfrak{m}}$	The expected processing time for one job $\mathfrak{j}_{c,\mathfrak{m}}$
$\mathfrak{P}_{\mathfrak{m}_i}$	Set of local transportation paths represented by jobs for moving one container locally between two intelligent devices $\mathfrak{m}_k \in \mathfrak{M}$

	and $\mathfrak{m}_j \in \mathfrak{M}$ at $\mathfrak{m}_i \in \mathfrak{M}$ such that a container c can be moved from $\mathfrak{m}_k \rightarrow \mathfrak{m}_i \rightarrow \mathfrak{m}_j$. Speaking, $\mathfrak{m}_i$ connects $\mathfrak{k}$ and $\mathfrak{j}$
post($\mathfrak{o}$)	The postcondition state an $\mathfrak{m} \in \mathfrak{M}$ is in after the operation $\mathfrak{o} \in \mathfrak{O}$ is finished
pre($\mathfrak{o}$)	A precondition state for an operation $\mathfrak{o} \in \mathfrak{O}$ before the operation can begin
$\phi_{\mathfrak{m},\mathfrak{o}}$	Processing time for one job $\mathfrak{o}$ on one intelligent device $\mathfrak{m}$
R_c	Temporary set of routes for one container c during route determination
$\rho_{c,\mathfrak{m}}$	The release date, expectation of arrival and ready for execution, of one job $\mathfrak{j}_{c,\mathfrak{m}}$
$\mathfrak{R}_\mathfrak{m}$	Local routing table as a set of 2-tuples of $(\mathfrak{d}, \tau)$ storing all possible reachable destinations and their previously evaluated times by pastCosts$(r_c, \mathfrak{m})$ at $\mathfrak{m}$
r_c	A timed trail (route) $(t_1, \ldots, t_i)$ for a container c representing its movement through the system
safe(r_i, r_j)	Function to check whether two routes $r_i \in R_i, i \in C$ and $r_j \in R_j, j \in C, R_i \cap R_j = \emptyset$ are deadlock-free
$\mathfrak{S}_\mathfrak{m}$	Scheduler organizing all jobs $\mathfrak{j}_{c,\mathfrak{m}}$ at one $\mathfrak{m}$ for container transportation
s	Message $s \in S_c$ seeking routes for a container c at a source $\mathfrak{m}_s \in \mathfrak{M}$, such that $s = (r_c, \mathfrak{d}, \tau, l)$, with $\mathfrak{d} \in \mathfrak{M}, \tau \in \{0, T\}$
S_c	Tree of seeking messages (communication graph) generated to compute r_c for container c in the system
seqTime($\mathfrak{l}$)	Function returning the time α a sequence of operations took to move one container c
σ	Experienced waiting time until a trail is found for c at source $\mathfrak{m}$

$S_{\mathfrak{m}}$	Set of states for $\mathfrak{m} \in \mathfrak{M}$ for transportation of containers
$\mathcal{SC}(c, \mathfrak{m}, \mathfrak{l})$	The static cost function returning the time τ as cost of transporting one container c by using a local sequence of operations $\mathfrak{l}$ at $\mathfrak{m}$
timeColl(r_i, r_j)	Checks whether a time collision exists between two trails r_i, r_j for container $i \in C, j \in C$ exists
timedSafe(r_c)	Check whether inverse flows from other $r_i, i \in C$ for $r_c, c \in C$ exists in the system
t	Timeslot $(e, \alpha, \beta), e \in E, \alpha \in \{0, ..., T\}, \beta \in \{0, ..., T\}$ representing one movement of c at an estimated start-time α and estimated end-time β
τ	Discrete time step
T	Last time step, such that $\tau \in \{0 \dots T\}$
$\mathfrak{t}$	The trail on which a message is sequentially forwarded between two intelligent devices $\mathfrak{m}_i$ and $\mathfrak{m}_j$, such that $t = (\mathfrak{m}_i, \dots, \mathfrak{m}_j)$
transTime$(\mathfrak{m}_i, \mathfrak{m}_j)$	Constant transmission time of one message between two intelligent devices $\mathfrak{m}_i$ and $\mathfrak{m}_j$
trans(t)	Transportation time trans$(t) = \beta - \alpha, s = (e_i, \alpha, \beta)$
opTime$(\mathfrak{o})$	The function opTime$(\mathfrak{o})$) returns the time of one sample $\delta \in \Delta$ to execute one operation $\mathfrak{o} \in \mathfrak{O}$ at one $\mathfrak{m} \in \mathfrak{M}$
$v_{\mathfrak{o}}$	Velocity of one $\mathfrak{o} \in \mathfrak{O}$ on which a container $c \in C$ has to be moved one $\mathfrak{m} \in \mathfrak{M}$
V_τ	Set of vertices at time τ
wait(s_i, s_j)	Waiting time between two time-slots t_i and t_j
w_i	Weight for edge e_i

1 Introduction

In today's globally acting supply chains, fully automatic transportation systems are necessary to overcome the amount of transporting goods worldwide. In 2014, the total air freight and mail volume were 14.3 million tonnes, with a growth of up to 40% for Denmark since 2013, and average growth of 6.4% for intra-EU and 3.3% for extra-EU transportation.[1] To overcome this amount of transportation volume, distribution centers are using more automation. In Germany alone, there was a production volume of 18.6 billion euro[2] in intralogistics[3] in 2013. In 2004 14% of the total production volume in Germany (that total being 12.8 billion euro) comprised IT software for machine control and communication only.[4] IT projects, in particular, tend not to meet the project schedules. Moreover, on average 45% run over budget and 7% over time. Even worse, if initially estimated with over $15 million, they massively surpass their budget.[5]

The above studies unsatisfactory results and the overall increase in transportation volume have led to a focused research and industry interest in automatizing IT software for transportation systems. Transportation systems are modularized on the machine level, which increases the reusability of one transportation unit considered isolated from the system. On a higher level, the material flow control, which is done by a routing method, is a highly complex project-specific solution based on the transportation system and client requirements.[6] However, a solution based on a "Plug and Play"[7] concept for decentrally organized transportation system is needed.[8] It provides the generalization of material flow control as well, such that project-specific implementations are deprecated

[1] See [Ee11, p. 117].
[2] See [Sta17].
[3] Intralogistics cover all local transportation systems. A more detailed explanation follows in section 2.1
[4] See [Arn06, p. 7].
[5] See [BBL12].
[6] See [GH10, p. 15].
[7] Also known as reconfigurable manufacturing system.
[8] See [GH10, p. 44].

and the system automatically recognizes the transportation structure and can operate on it. This leads to reductions in software construction and -commissioning, and provides stability to meet the project deadlines, in contrast to the results of the studies mentioned above. This material flow control needs to adopt advantages in scalability and flexibility[9] and provide robustness[10] for industrial automation to prevent blocking and deadlocks during transportation. By adopting the "Plug and Play" idea, the material flow methods must be able to react to system changes and still provide the possibilities of deadlock-free transportation.[11]

All these restrictions are being affected by the modules of the material flow control: system structure for operation; evaluation of the transportation unit; and path determination, selection, and execution. Since all these modules operate in a decentralized environment, no global state knowledge is present. Every task within the control has to be coordinated with other transportation units within the system. Otherwise, blocking or a deadlock occurs, and a system break-down is unpreventable. Therefore, it is clear that developing a method for material flow control in a decentralized environment is a complex task and, as previously mentioned, a project-specific solution results in a project failure in time and or in the budget. A fully automatic and project-independent solution that recognizes the system structure can operate automatically after manual installation, minimizes the time spend on software development and commissioning, and significantly reduces the risk of project failure.

1.1 Purpose of this work

This work develops a routing method for material flow control for automated decentralized structured complex transportation systems under the restrictions of it being scalable, flexible and robust with regard to an intralogistics supplier. Fully automated containers have to be transported between the source- and destination hand over units. The method has to provide, after an initial system recognition, fully automated transportation. It has to scale linearly with the number of used transportation units and must be flexible by

[9] See [Vya13, pl. 1237].
[10] See [Arn06, p.225].
[11] See [FSG10].

properly reacting if a unit is being added[12] or removed[13] from the system. In addition, the method is called robust if no deadlock occurs during the execution of the transportation itself, and blocking has to be minimized.

These restrictions - scalability, flexibility, and robustness - ensure a reduction in software development and later commissioning for transportation systems for an intralogistics system supplier in Germany. This supplier‘s specialization is the planning, design, manufacturing, installation, and commissioning of tailor-made transportation systems. Based on standardized transportation units, systems are planned and individually designed to reflect the client‘s needs. This individualization is a specific system design, adapting the transportation unit design and amount to assembly. Since on a low level of control, the transportation units are well defined and reusable[14], the system design is still developed manually and in relation to the project. Especially the material flow control differs from project to project; therefore, the goal of this work is to provide routing for material flow control for complex[15] transportation systems. To evaluate the method, a real-life system under planning is used. It consists of several thousand transportation units for one of the world‘s leading parcel and expresses delivery companies. In an agreement, all used data have been anonymized.

All in all, this work develops a method that enables scalable, flexible, and robust routing by decentralizing the computational overhead by optimizing the communication on a low level at each machine. This enables computation on a small scale by distributing complex tasks to a defined number of computational units, such that in a collective manner a solution, here a robust transportation path, can be determined and subsequently executed.

1.2 Structure of this work

This work is structured as follows: Section 2 explains the domain of automatic transportation, along with interfaces and their subdomains. The reader is given a broad overview

[12]This could be by extending the system with additional transportation unit in the big scale or in small scale by repairing a unit.

[13]E.g. break down of a transportation unit, shutdown for maintenance, etc.

[14]A change in length or capability in handling special kind of container types does not affect significantly the low level control of motors.

[15]Complex in the sense of size, number of transportation units, and considering changes over time, e.g. break-down, extending and reducing the system structure.

of material handling at machine-, low-, and high-level control. Standardized machines[16] are introduced as an example to form a transportation system. Low-level control is needed to perform local actions by controlling the motor by parametrization based on sensor data. Later, the section introduces high-level control, which is responsible for the material flow in intralogistics systems. Broadly, the section covers high structures such as enterprise resource planning components to introduced its interfaces to the material flow level. Finally, an overview of other domains related to transportation, such as graph routing, is provided to complete the picture.

Section 3 presents an overview of current existing methods with a focus on transportation systems, in particular, the domain defined in the previous section with its specific characteristics. To broaden the discussion, domains with similar or close characteristics are considered too. The methods in those broader domains are being explained and later used for concept finding to solve the issues and tasks explained in section 2.

Subsequently, section 4 summarizes the open topics of interest to solve the issues and tasks presented in section 2. Then, section 5 first introduces the protocol used as a basis for the method. The section then discusses the local behavior of one transportation unit depending on every functionality of the protocol, followed by the routing method responsible for the material flow. Next, section 6 explains the implementation of the methods and the simulation environment and section 7 introduces the experimental design and outcome. Finally, section 8 concludes the work and proposes open points for future research to optimize the introduced methods.

[16] Later called transportation units and positional units.

2 Problem description

To provide a deeper insight into the object of study, this section is split into three parts. First, section 2.1 presents the basic terminology to ensure an understanding of the domain of logistics, in particular, "intralogistics" and its highly complex material flows providing local transportation. Afterwards, section 2.2 covers the resulting specialties, which lead to system boundaries in routing within this domain. Finally, section 2.3 describes the resulting challenges, which lead to specific issues that must be addressed to provide a feasible method.

2.1 System description

No unified, world-wide accepted definition of logistics[17] exists. One short description is provided by Rushton et al.:[18]

> Logistics = Materials management + Distribution

For a broader scope, the author defines supply chain as:

> Supply Chain = Supplier + Logistics + Customers

A more detailed definition of logistic by Rushton et al.[19] is:

> ... the efficient transfer of goods from the source of supply through the place of manufacture to the point of consumption in a cost-effective way whilst providing an acceptable service to the customer.

[17] See generally [AIK+09, p. 3], [RCB17, pp. 4–5], [GBS13, p. 2]. Commonly, the definitions distinguish themselves by additionally restrictions to specific logistic oriented systems or objects according to [AIK+09, p. 4].

[18] See [RCB17, p. 4].

[19] See ibid., p. 6.

Intralogistic

Both the logistics and supply chain, are concerned with the physical and information flows from the raw material to the final distribution of the finished product. Material- and supply management considering the storage and flows into and through the production. On the other hand, distribution considers the storage and transportation of the final good to the customer.[20] Also, Arnold et al.[21] defines the storage and flows as material flow, whereas transportation in the material- and supply management or within distribution centers. Material flow is defined as the structured and organized movement between two points considering the efficient use of space and cost as energy and personal savings.[22] This work uses Tempelmeier's[23] definition of material-logistics for both material and supply management as summarized. To further distinguish the material flow within the supply chain management, Arnold[24] introduces the term "intralogistics"[25] as follows:

> "Die Intralogistik umfasst die Organisation, Steuerung, Durchführung und Optimierung des innerbetrieblichen Materialflusses, der Informationsströme sowie des Warenumschlags in Industrie Handel und öffentlichen Einrichtungen."

which can be translated into:

> "Intralogistics covers the organization, control, execution and optimization of the internal material flow, information flow, as well as the turnaround of goods in industry, commerce, and public societies."

Material-Flow

In this work, it is assumed that the production beforehand and the final customer delivery are optimized and not considered. Therefore, the focus is on internal material flow only. Thus, Gudehus[26] makes an abstraction based on source, sink, stations, and connection with the terminology (n, m), with n as the amount of input and m as the amount of output of one elemental station[27]. This leads to the following used definitions:

[20] See [RCB17, p. 4].
[21] See [AIK+09, pp. 4–5].
[22] See [MS02].
[23] See [Tem05].
[24] See [Arn06, p. 1].
[25] Gudehus (see [Gud13, p. 550])
[26] See ibid.
[27] An elementstation, short station, is one transportation node or element within the transportation network, e.g. conveyor.

- Source: A station $(0, m)$ whereas entry flows are not considered.[28]
- Sink: A station $(n, 0)$ whereas exit flows are not considered.[29]
- Station: (n, m) with n entry and m exit flows.[30]
- Transportation connection: a connection[31] for transporting goods that are moved without stop; the only possibilities for a stop are goods with higher transportation priority, backlog, or malfunction.[32]

Based on these definitions a logistic network[33] can be formed, e.g. as a graph[34] with vertices and edges, representing stations and connections. This work uses the term transportation unit (TU) instead of station. All stations in this work are TUs capable of moving goods from a source s to a destination d. In [Gud13, pp. 457-475], stations are also working places.

Complex networks are a specific classification of networks. They are distinguished from regular networks by their irregularity, complexity, and evolution over time,[35] such that edges and vertices are added and removed over time.[36] Adapting this network definition to transportation networks it leads to the possible adding and removing[37] of elements, e.g. conveyors. Such a dynamic in networks can be caused by machinery failures such as break-down of one conveyor in small scale, or of a part of the network on a large scale.

Fig. 2.1 shows possible flows of goods caused by transportation. Gudehus introduces in [Gud13, pp. 463-472], the continuous, discontinuous, forking and joining capable connections. Continuous connections are TUs able to move a good from the entrance to the exit without stopping. They only stop when a backlog or malfunction occurs. Such units are conveyors, e.g. belt-, roller- and skid conveyors. Discontinuous connections are units that transport capacity of goods separately before executing the next transportation. This separation of the transportation task is the result of a specific mechanical/electronic

[28] See [Gud13, p. 459].
[29] See ibid., p. 460.
[30] See ibid., p. 457.
[31] For sake of simplicity, by the use of connection a transportation connection is meant.
[32] See [Gud13, pp. 462–463].
[33] A logistic network is an amount of sources and sinks which are connected by connections. (see ibid., p. 550)
[34] See sec. 3.2.
[35] See [BLM+06, p. 177].
[36] See [PRD03, p.2].
[37] Adding and/or removing will be summarized as dynamic in network.

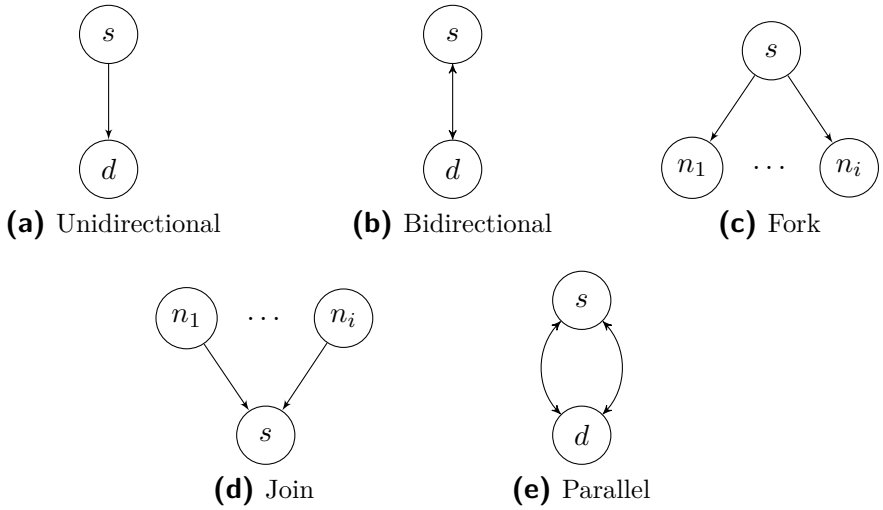


Figure 2.1: Possible flows of goods in a transportation system represented by graphs.

condition. An example is a right angle deck with the terminology of $(1, 1)$, where a goods direction of transportation will be changed by, e.g. lifting it and then moving it by 90 degrees. More complex discontinuous transportation units with (n, m) enable the forking (fig. 2.1c) and joining (fig. 2.1d) of connections. A right angle deck can be extended by additional connections, its operational basis increases by distributing container in several connections.

In addition to Gudehus‘s work, three further subdivisions are used in this work: all transportation units can transport unidirectionally, bidirectionally, or in parallel. These three additional possibilities allow more detailed planning to find additionally paths. Unidirectional transportation (fig. 2.1a) allows the movement of goods only in one previously[38] defined direction. On the other hand, bidirectional transportation (fig. 2.1b) allows the transportation of a good in the opposite direction respectively in time[39]. Here, blocking or deadlock can occur if two connected conveyors transport in opposite directions. Then, in a worst-case scenario, two goods would collide, or at least blocked. Parallel transportation (fig. 2.1e) occurs when more than one transportation route exist between two TUs.

In the transportation of containers, fluid transportation is intended to reduce the transportation time. The overall amount of blocking should be reduced; blocking leads to a higher transportation time, and further could result in deadlocks. Blocking occurs

[38] e.g. during system planning.

[39] One example are conveyors which can change their direction in transport to forward or backwards.

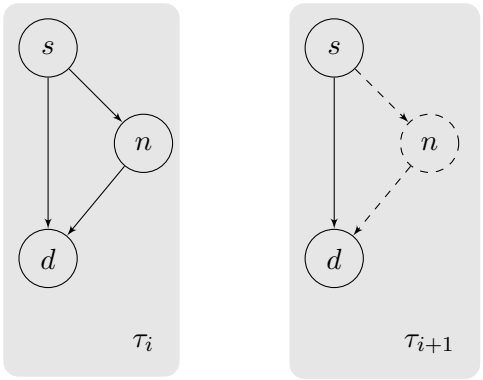


Figure 2.2: Example of a complex network evolving over time from τ_i to τ_{i+1} by a breakdown of n.

when one container's travel path is being blocked by another container. Blocking is the stumbling of another container such that a detour has to be considered before the transportation path planning or online during transportation execution.

As mentioned previously, transportation systems are more complex. During run-time, TUs can break-down or specific parts of the system have to be shut down. Fig. 2.2 shows a complex graph over time. At time τ_i two parallel routes exists to the destination d. The TU n has to be shut down and the flows (s, n) and (n, d) are no longer possible. Also, in complex networks with time slots a transportation unit s status from a source can be switched to a destination. Therefore, respectively to a time τ_k s can switch its roles.

A routing method[40] consists of three steps:[41] a path request, a path determination, and selection. Path execution must also be considered. Path request is the initial notification that a path is needed from s to d. The determination requires the possibility to compare TUs for later selection. Many comparison methods have been used in the literature, but a common is a cost model. Costs are not restricted to monetary costs, e.g. operational costs, other variables are considered too, e.g. distance, transportation speed or transportation quality, and fault probability. New models consider a dynamic change in costs for a model to represent the changes over time of one running system. Furthermore, the routing, is the path selection based on the previously evaluated TUs with the given cost model. If more than one path exists from s to d, an evaluation must be conducted based on these paths to select the final path for execution. A dynamic

[40] Also called protocol.
[41] See [Cha15, p.3].

model allows load balancing during runtime. This ensures that one specific region is not overloaded with transportation execution if a static or insufficient dynamic model defines it as the fastest[42] one. Finally, the execution of the transportation during the determined path must ensure a synchronization such that no blocking occurs. Especially in decentrally organized environments without a centralized unit knowing the global state, each TU must communicate with its neighbors and exchange statuses and future steps.

All these steps rely on synchronization of TUs. Therefore, a unified communication is needed at each unit. During each step, information has to be shared for state information collection and decisions that have been or will be made. Each network has its communication limitation given by the bandwidth[43]. When the bandwidth is exceeded, all additional data are delayed in arriving at the destination. The result is crucial and in industrial applications dangerous, particularly when a hard real-time system is given, as described later in section 2.1. Two data types need to be shared: first, status collecting data for decision making; and second, the results of made decisions. The first type is also called information collecting. A minimum is a simple check of whether a TU n can reach a destination d. To improve the flows within one system the cost model additionally has to be applied to n to enable comparability of units. Secondly, if a decision has been made, e.g. units have been dropped because of the cost comparability, this information has to be shared by informing other units using a pull[44] or push[45] functions.

Current Industrial Systems

The majority of current material flow systems are structured and controlled in a centralized way, by few servers collecting world state information and computing, based on this information, the next steps within the material flow.[46] Fig. 2.3 shows the structural layers to enable material flow in a centralized manner. The lowest level, field

[42] Fast in terms of throughput. [DSL14] reveals that even methods able to handle dynamic environments inspired by biologic tend to overload a specific region caused by a static cost model or by insufficient load balancing model.

[43] Bandwidth in the domain of computer science. It refers to the data rate measured in bits/sec. In contrast, for electrical engineers bandwidth describes the quantity in Hz, see [TW13, p. 93]. Here, the definition for computer science of data rate is used.

[44] Request an information by the unit itself. When requesting it cannot be guaranteed that the information exists.

[45] Other units will be updated automatically at bandwidth expense.

[46] See [GH10, p. 17].

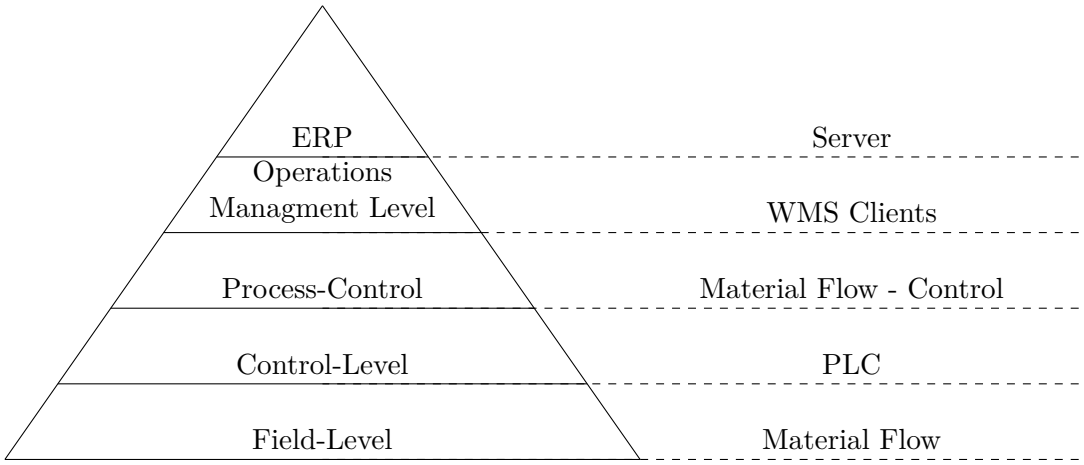


Figure 2.3: Simplified controls-pyramid for centralized material-flow systems (cf. [GH10], p.17)

level, covers the machinery actions and information collecting of sensors. It can be imagined as a pipeline from the control level to the direct machine to execute defined orders, whether continuous[47] or discontinuous[48]. The next level transforms the high-level orders into a machine-understandable format and the sensor data for higher levels. Distributed control cabinets including the programmable logic controllers (PLCs)[49] with their extensions[50]. Also, the PLC is responsible for safety and security functions.[51] This level communicates with the next higher level, the process control, which is responsible for the material flow within the system. Process control coordinates machine movement based on understandable tasks for the PLC. Above this level the operations management level coordinates tasks that will be organized and executed by the material flow, e.g. commissioning, breaking and building of containers. Finally, the highest level, the enterprise resource planning (ERP), represents the higher-order business processes.[52] An example of a hardware configuration is given in fig. 2.4. Controls are responsible for machine behavior by parameterized speed and directions for converters. The speed of motors, status and other system relevant statuses are determined by sensors. Both converters and sensors can be distributed within the system. Therefore, Remote I/Os are used to increase the accessibility range of PLCs. PLCs and human-machine-interfaces

[47] A continuous task will be executed until a hold/stop order is sent.

[48] Discontinuous tasks are specific timely defined tasks, e.g. start and end time.

[49] A PLC (Prommable logic controller) is a special microcontroller storing instructions and functions, e.g. logic, sequencing and timing, to control machines and processes. (see [Bol09, p.3])

[50] e.g. digital / analog converters or communication units.

[51] See [GH10, p.17].

[52] See ibid., pp. 16-18.

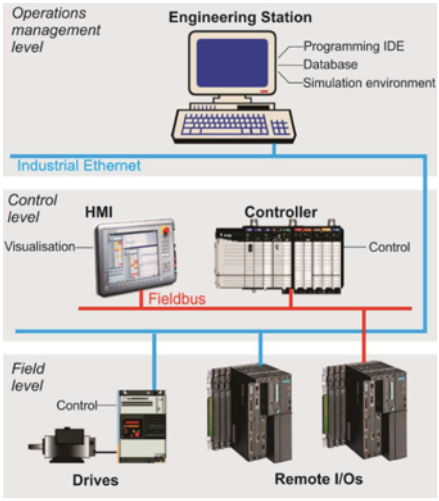


Figure 2.4: Hardware architecture at factory floor automation systems (see [Vya13], p. 1235)

(HMIs) are used for low-level control; visualization and interaction are done by the HMI. One level above, the server with further system control is installed and running, e.g. material flow software, for control and optimization.

Centralized structured systems use project-specific optimization. Because all information is collected in a centralized way and is specifically configured[53]. A design change in the layout of the material flow has the consequence that all layers in the controls pyramid have to be adapted.[54]

[53] In PLCs, each input and output to an hardware, e.g. motor or sensor, is hard coded in the Remote I/Os (Input / Output) reference table connecting one input or output at a PLC with a real-world sensor or motor. This hard coding is tested while checking the commissioning phase of each I/O to determine whether the correct hardware is being hard-wired.

[54] See [GH10, p. 19].

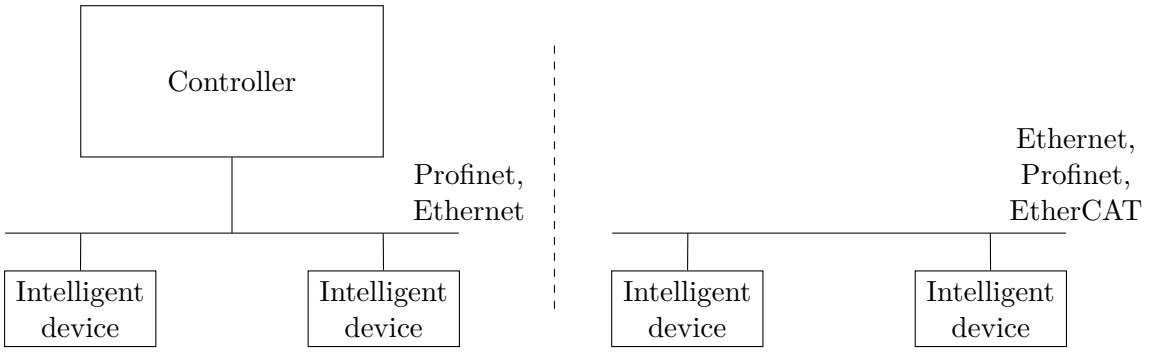


Figure 2.5: Comparison of current centralized systems (left side) and decentralized systems (right side) (cf. [GH10], p.24)

Decentralized structured systems

Current design requirements for manufacturing and logistic systems are rapid reconfiguration[55], easy integration[56] and hot backup redundancy[57].[58] To manage and meet these requirements inside and outside of an assembly system, several concepts have been introduced. The most famous are the flexible manufacturing (see, e.g. [JKB03]) and reconfigurable manufacturing (see, e.g. [KHJM99]) for the hardware part, and holonic (see, e.g. [WO05]) and biological manufacturing (see, e.g. [UVO97]) systems on the software side. All systems share the same ideology, subdividing the assembly systems into stations, cells, buffers and transport systems in subsystems. These subsystems and modules received defined degree of freedom in autonomous.[59] Fig. 2.5 shows the current structure of a decentralized structured system with intelligent devices compared to a centralized system.

From the hardware perspective, reconfigurable manufacturing is considered as the future paradigm.[60] Since flexible manufacturing only considers software reconfiguration on

[55]Also known as automatic reconfiguration, it describes the possibility of performing proactive or reactive adaptation to specific processes as stated by [WHM13, p. 82]. In [RSAV16, p. 133] the concept of reconfiguration is split into connection, communication and interaction of loosely coupled system units, e.g. conveyors, which have to be organized to provide a system functionality as transportation.

[56]Integration in terms of enabling the possibility by implementing control of decentrally organized systems. Industrial standard programmable languages have to be used as IEC 61131-1 (see [JT10]) or its successor the event-driven architecture defined in IEC 61499 (see [Thr06, pp. 115]) which covers interoperability, portability and reconfiguration as stated by [DV10, p. 1]

[57]A hot backup is a fully duplicated backup. Current industrial techniques is backup-in-depth , see [Com14, p. 5-18].

[58]See [DV12, p. 390].

[59]See [SF07, p.712].

[60]See [DTOJ14, p. 1].

an existing hardware basis, the reconfigurable manufacturing focuses on exchangeable hardware as well.[61] Intelligent devices are prior capable of reconfiguring themselves[62] such that a valid network exists for future decentralized operations and methods. Such devices are field devices which include a part of the control logic from the controller. This enables processing of signals directly at the place where they occur.[63] Intelligent devices communicate with each other if higher-level[64] decisions have been made. To make a higher level decision, each intelligent device is composed of several levels within the controls pyramid.[65] The level of composition is determined by the modules and its tasks. When considering a composition from field level to process control, the motors, sensors and at least one PLC is required. As mentioned earlier, in a centralized structure the motors and sensor are directly interacting with the environment, whereas the PLC is directly mounted to the intelligent device. It is hard-wired with the sensors and motors without bridging a certain geological distance. Up to a certain degree, each device is capable of decision-making without interacting with other devices, e.g. status reporting. For more complex tasks such as material flow, however, the devices have to communicate to find a sophisticated solution[66].

Communication is crucial to enable a proper material flow with intelligent devices operating in decentralized structured systems. To determine a better solution, each must share their current status in a real-time computer system manner[67]. Further, real-time is being classified as hard or soft real-time. If a system must meet at least one hard deadline[68], then it is called a hard real-time system or safety-critical real-time computer system.[69] Since missing a deadline in transportation systems causes damage of goods,

[61] See [ElM08, p.4].

[62] For reconfiguration of manufacturing systems several higher-level methods enable communication to provide higher lever methods to operate in a decentralized manner. For example [DTOJ14] proposes a service oriented architecture for reconfiguration of manufacturing systems, [ABH+13] presents reconfiguration at the network level, [OLBH12] introduce new self-recognizing drivers, [TKHP02] focus on identifying automatic field devices, and [SMA+03] introduce a multi-layer for establishing communication and interaction between robots in manufacturing systems.

[63] See [GH10, p. 24].

[64] An higher level is one of the possible levels above the field-level in the controls-pyramid from fig. 2.3

[65] See [GH10, p. 25].

[66] A sophisticated solution could be in means of transportation time, e.g. minimizing the time to move one container from s to d.

[67] A real-time computer system is also a computer system (analogous to a PLC, which is a computer system as well) that depends on the correctness of the computations in logical and physical on time. (see [Kop11, p. 2]).

[68] When the consequences of missing the deadline is sever, the system is called hard.

[69] See [Kop11, p. 3].

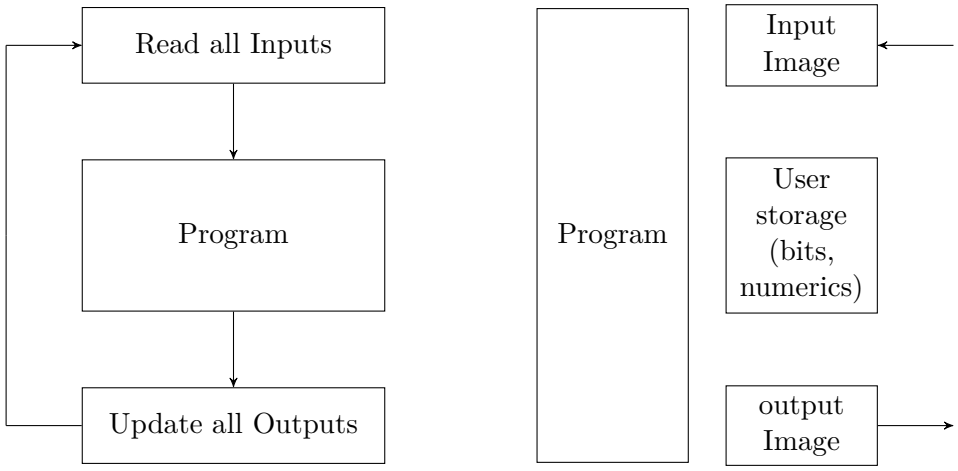


Figure 2.6: The program and memory organization of a PLC, see [ML85, p. 16.12].

or in the worst case harm to employees, those systems are classified as hard real-time systems[70].

Structure of intelligent devices

PLCs are industrially accepted intelligent devices.[71] The left side of fig. 2.6 displays a PLC's action. One action is called a program scan. At its start, all inputs signals are read and its status transferred to the internal input-memory. Based on the signal's value within the memory, the program is executed, and its solutions for each output signal is stored inside the output memory. At the end of the scan, the output-memory is mapped into all outputs.[72] As shown in fig. 2.6 on the right side, the memory of the PLC is split into four areas. The input and output memory areas mimic the input and output signals, respectively. Furthermore, there is a restricted memory area that is not connected to the outside, e.g. storage bits, timers, counters. Finally, the last area is the place where the program itself is situated.[73] This structure is used in classic PLCs and soft-PLCs[74]. A PLC is based on a special defined programming language defined in IEC 61499[75] and simple data-structures.[76] Several additional functions are being provided, e.g. counters,

[70] A more detailed classification can be found in [Kop11], p. 14.
[71] See [GH10, p. 25].
[72] See [ML85, p. 16.11].
[73] See ibid., pp. 16.11-16.12.
[74] Soft-PLCs are industrial computers (IPCs) with a specific restricted computational area, in computational unit as well as memory, providing classic PLCs functionalities.
[75] See [Thr06, p. 115].
[76] See [Arn06, p.217].

timers, etc., as well as communication units offering an interface to other units within the system.

Communication in decentralized systems

Since each intelligent device[77] has to make decisions that affect other intelligent devices in a decentralized system, system state information has to be collected, decisions of other TUs must be considered, and own decisions need to be spread across the system. Firstly, on the field-bus level, as the connection between the PLC and the sensors and motors, the Ethernet[78] standard could not be established. This is because it was designed with different Quality of Service (QoS) since a high determinism and real-time data transfer are needed compared to commercial networks.[79] Therefore, Profibus[80]is still mostly used for communication between a PLC, sensors and motors.[81] Since existing communication methods in industrial automation were developed two decades ago, they lack data throughput. There was a need to upgrade speed and performance of existing non-real-time networks, mainly Ethernet networks.[82] Today, communication in industrial networks is being classified in three categories, which are differentiated by network update times[83]. To provide a reduced update time, the regular Ethernet MAC - layers[84] must be interfered with. Fig. 2.7 shows the network stack configuration. On the left, the open system interconnection (OSI) reference model is shown. It was developed in 1983 as a first step to international standardization of protocols defined in difference layers for open communication with other systems.[85] Each layer is responsible for the following layer, as defined by Tanenbaum in [TW13, pp. 43-45]:

[77] One intelligent device is mounted on one TU.

[78] Ethernet is a group of standards for network communication for local networks as well as for huge networks and is standardized by Institute for Electrical and Electronics Engineers (IEEE), see [Spu00, p. 11].

[79] See [G013, p.876].

[80] Defined in [DIN11, pp. 118-201], which is a communication standard for exchanging data between devices.

[81] See [TKHP02, p.143].

[82] See [Dec05, p. 1102].

[83] Scheduled information transferred with one rotation within the network. It consists of scheduled traffic, unscheduled traffic and maintenance traffic, see [Dun05, pp. 369-370]

[84] MAC - Layers define the protocols accessing the Ethernet system, see [Spu00, p. 18].

[85] See [TW13, pp. 41-45].

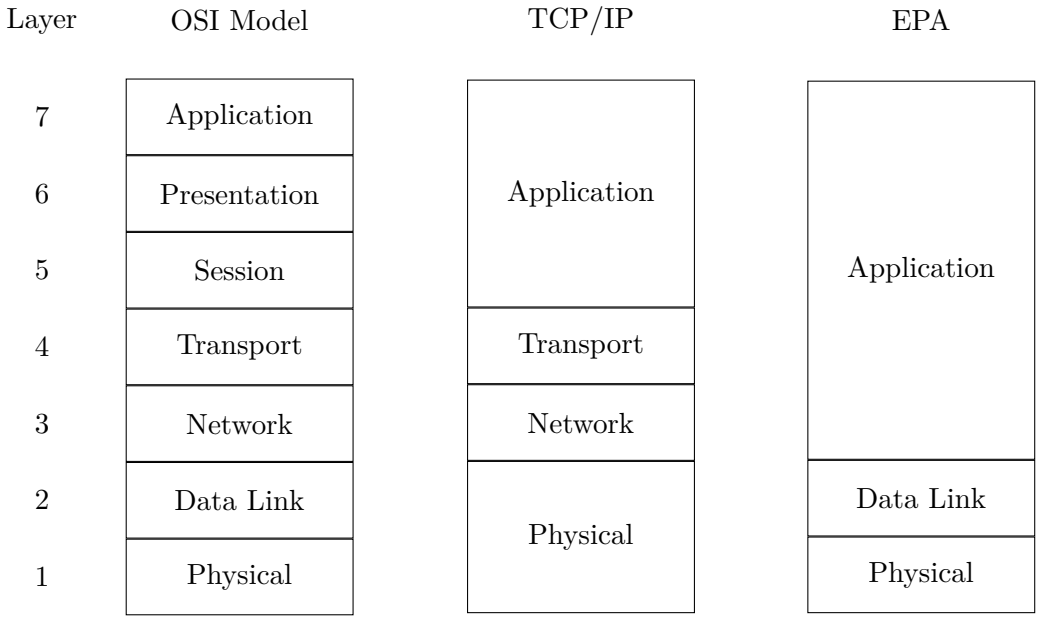


Figure 2.7: Differences between the network stack configuration of conventional networks and industrial networks, see [G013, p. 868].

- The physical layer: This layer is responsible for ensuring that bits are sent and received as the correct value and in the right order by an electrical signal representation.
- The data link layer: Its main task is to transform a raw transmission into a stream free of errors.
- The network layer: It determines the routes for packets from device s to d.
- The transport layer: In this layer, received packets are accepted or rejected as well as split into smaller units if needed.
- The session layer: This layer enables the establishment of different sessions (e.g. dialog control, token management, synchronization, ...) between different machines.
- The presentation layer: Here the layer is concerned with the syntax and semantics of the transmitted information.
- The application layer: A variety of protocols needed and used by the users/device are containing within this layer, e.g. Hyper Text Transfer Protocol (HTTP).

Most communication protocols do not face all the problems defined in the OSI-reference model, such as the reduced model TCP/IP (see fig. 2.7 center). It merges several layers

into one for the sake of simplicity; this layer is still fully functional, widely used in the Internet, and is the basis of real-time Ethernet.[86] Similar to the TCP/IP model, enhanced performance architecture (EPA)[87] is a reduced layer model; it consists of three layers only (fig. 2.7 right side), one more less than TCP/IP and is still functional. All missing layers are merged with existing layers, to speed up the throughput of communication. Therefore, merging of layers is based on the specific system, e.g. the size of the application layer in relation to data link layer is depending on the system.[88]

2.2 Requirements description for routing

As mentioned previously, decentrally organized systems lack the possibility of one centralized unit with global state knowledge organizing each TU distributed in the transportation network. This leads to decentralized TUs being controlling by themselves and therefore relying on correct and on-time information for decision making. This results in local self-management to reach a global goal: the transportation of containers through the transportation network.

Based on the described domain in section 2.1, the following describes the practical requirements with regard to an intralogistics system supplier, a worldwide turnkey supplier in the domains of air cargo, automatic car parking and industrial transportation systems. All three domains include complex transportation systems for moving goods. They depend on safe and on-time transportation to fulfill the given time requirements. Whereas, unit load device (ULD)[89] in air cargo terminals, cars in automatic car parking systems, and materials in industrial transportation systems all require a transportation system with high availability[90] and a predefined throughput requirement to reduce waiting times for goods.

[86] See [G013, p.869].

[87] Developed in MAP-project (Manufacturing Automation Protocol) by General Motors and Boeing as a standard communication protocol within CIM (computer integrated manufacturing) hierarchy, for a detailed history background see [G013, pp. 860-876].

[88] See [Pat98, p.279].

[89] ULDs are loading units for aircraft's, see [AIK+09, p. 763].

[90] Availability of the system in terms of ability to provide specific operations in predefined conditions failure-free over time, see [AIK+09, pp. 865-869].

2.2.1 Segmentation of the requirements

Since hierarchical structures are currently being used for transportation systems, centralized monolith software has to be written to coordinate the material flow. This is due to project specific transportation systems resulting in a new project specific transportation network representation on which algorithms operate on to find proper routes. Practically, this material flow software differs from project to project, resulting in new software development in each project realization phase, which later needs testing in the commissioning phase as well. In addition, adaptions to the material handling system are made when requirements change, due to the need to adapt to the market. Then, as Furmans et al. stated in [FSG10], current automatic material handling systems are inflexible; changes are cumbersome and expensive once the system is installed. As mentioned earlier, the intralogistics system supplier being used as a reference is specialized in complex material flow systems. Complexity arises from the number of sources, sinks, and their interconnections given by equipment. Still, according to Mayer,[91] a time-discrete routing for such complex systems for reconfigurable manufacturing [92] is missing. Even though this seems outdated, still current systems are not able to provide sufficient solutions for such complex material flow systems, as shown in chapter 3.

Vyatkin[93] points out that the modularity, integrability, flexibility, scalability, convertibility and diagnosability are main concerns for reconfigurable manufacturing systems. In addition, El Maraghy[94] states that routing has to be mainly flexible and reconfigurable to switch and react to changes. Moreover, a major concern for industries is robustness too, as stated by Arnold.[95] Therefore, this work will focus on developing a routing method to provide material flow in complex intralogistics decentrally organized systems[96] with the following attributes:

- scalability: handling of complexity as the number of components arises.
- flexibility: the capability of adding and removing components.
- robustness: ensuring system stability during transportation.

[91] See [May11, p. 75].
[92] This work only focuses on transportation systems within reconfigurable manufacturing.
[93] See [Vya13, p. 1237].
[94] See [ElM08, p. 13].
[95] See [Arn06, p. 225].
[96] For the sake of simplicity, in the following this is referred to complex intralogistics systems unless otherwise stated.

Analogues, these are the major concerns in current research in routing according to Boukerche,[97] Chakchouk,[98] Guenther,[99] and Furmans.[100] Scholz-Reiter and Freitag[101] also refer to self-adaption, self-optimization and self-organization. This work focuses on self-organization only in so-called autonomous manufacturing systems[102]. Interest is in an intralogistics system supplier who operates in several areas; however, only air-cargo and automatic carpark material flows are considered. Several prestige automatically car park systems have been realized, from 300 to 1,000 fully automatically handled parking spaces. In the domain of air cargo, one of the world's largest fully automatic cargo-terminal has been planned and realized. Here, material flows of standardized equipment have been consolidated into one system. Still, a major part is the software development regarding material flow. Therefore, the main concern is the reduction of effort in development of material flow systems. Since the field-level has been standardized, the next level, the process-level, is currently under investigation to be automated as well.

2.2.2 Requirements for routing

The main requirement for routing in complex intralogistics systems can be defined as follows: a container c[103] has to be moved from s to d, where, the arrival of c at s is ensured by an upper-level control[104]. To provide transportation, each component on the hardware and software side in one system, have to operate together. Since a PLC operates in a hard real-time environment, the routing method has to cope with restrictions too. These are fixed cycles times that have to be ensured, and a restriction in data management, as described in the subsequent paragraphs.

[97] See [BD15].
[98] See [Cha15].
[99] See [GH10].
[100] See [FSG10].
[101] See [SF07, p.725].
[102] Scholz-Reiter and Freitag describe this kind of manufacturing system as the next generation system after reconfigurable manufacturing system. The difference between these systems relies in an external force used for decision making, which other authors still see as reconfigurable manufacturing system, since here the external force can be an computer program as well, e.g. ERP system.
[103] In reference to the air cargo domain, containers are the goods that are being moved within an transportation system.
[104] This could be a warehouse management system to define s and d and a guidance system, e.g. forklift-guidance system, to ensure the arrival of c at s at a specific time, which is out of the scope of this work.

Scalability

Individual solutions for container transportation which consisting of a large number of TUs are cope of complex transportation systems. In such systems, a routing path has to be found in a proper amount of time[105] to avoid congestion at the source due to the container arriving at the source s for routing to the destination d before the routing method determines the path. High scalability of the routing method allows the supplier to provide larger systems with a reduced amount of software development and commissioning. Since, in industry, project-specific routing based on the system characteristics is developed; this software has to be programmed and tested in the commission phase for its running capability before handing over. In decentrally controlled environments, each TU is equipped with an intelligent device able to compute an action, based on retrieved or stored information. As previously shown in fig. 2.6 previously, scalability and additionally determinism are important. Since each program is executed between the reading of inputs and writing of outputs, its execution time affects the reaction time of transportation units too. More critically, if safety functions[106] are also executed on the same PLC as well, several safety standards[107] have to be ensured. Finally, a TU and the system need a CE certification[108] separately, to be allowed for operation.

The described restrictions in the design and operation of TUs affect the possible computation time of the routing method. Independent of the size of the system, the method has to find a proper path in an acceptable time. Therefore, the method cannot use excessive algorithms to compute a path and or sub-path if it blocks the PLC cycle for too long. Since a safety function should be executed in a cycle first, the blocking of a complex[109] algorithm would result in an input skip[110]. In hard real-time environments, the execution of a (part of a) program must be ensured to be finished within the defined cycles times to enable the next execution of the cycle, including reading inputs and writing outputs.

[105] At least before the container arrives at the source.

[106] E.g. emergency stop when a specific point has been passed that could be hazardous for the building structure, containers move to a point they should not, such as walls, building columns, etc. It is more hazardous if a TU‘s execution is connected to safety equipment such as lightning barriers which ensure the immediate stopping if a person enters an automatic operating system.

[107] See ISO 12100 in [Sta10] defining design possibilities and risk assessment of machines overall including TUs.

[108] See machine directive 2006/42/EG in [Uni06] defines safety aspects a machine has to fulfill before operation and is a basis for the CE certification as defined in the regulation 765/2008 of the European Parliament in [Uni08].

[109] Complex in terms of time.

[110] When in a program execution the input value changes,e.g. bit-flip from 0 to 1 and then 1 to 0, before the program has been executed. These bit-flips would never be recognized by the PLC, since due to the excessive algorithm the cycle time is extended.

Furthermore, restrictions based on the DIN EN 61499-1[111] apply. The most restricting one is the possibility of allocating memory for storing data. To keep determinism in control logic, the allocation of dynamic arrays and pointers is impossible. If arrays or pointers are used, they have to have a fixed data size, and a pointer has to point to a fix data in memory. To overcome this restriction soft-PLCs[112] have been introduced. A query for a dynamic allocable memory is an asynchronous communication to the non-real-time executing part of the soft-PLC. Therefore, a result cannot be expected in the same cycle which the query for the information is triggered.

Besides the restriction in asynchronous communication within one soft-PLC, restrictions to the global communication also apply, given by a decentralized environment with distributed intelligent devices operating transportation units. First, no global knowledge exists. In a hierarchical architecture, one unit that is powerful in computation and sufficient memory administrates all information and the orders of each TU. Current existing PLCs cannot cope with the huge amount of data given by the collected information from each device. Therefore, expensive industrial servers are being used in this kind of architecture. They execute monolith programs developed and parameterized for one specific project. Here, in a decentralized environment with a heterarchy, each device has to collect information in a period of time to compute each step, whereas the synchronized transportation of a container requires status exchange between intelligent devices to compute a feasible path for routing. Even if a device requests all possible information, it has to cope with the blurriness of time, invariant of the requested data. Any information is not being received instantaneously, network errors, such as jamming[113] and dropping can occur in industrial networks. A resending of the data results in a time delay. Several computation cycles could already be passed, and therefore, the requested data could be outdated. If it was requested before the state change of a unit, the new state could be unknown to the current devices which are computing the next actions.

Finally, the intelligent device faces the execution of a computed path. Since no central device exists to control the current device and its TU, it has to store and administrate its own execution plans. To increase the throughput, the devices have to operate together in

[111] See [DIN05].

[112] See [Dun05] pp.94-96. A soft-PLCs is a personal or industrial computer with the capability of reading and writing I/Os from machines. To enable real-time capabilities in a multi-core CPU system, one CPU only executes the PLC-controller logic only, while the other CPUs operate the other non-real-time methods.

[113] Synonym for blocking in the network domain.

a synchronized way. A stop-and-go solution[114] would dramatically reduce the throughput, since a container would be stopped before a synchronization with the next TU happens.

In summary, scalability of intelligent devices for routing faces several issues:

- Hard real-time capabilities in cycle execution during run-time.
- Computational and data storage restriction for soft-PLCs.
- Collection of other devices state for decision-making.
- Execution of transportation and synchronization with other devices for fluid transportation.

Flexibility

Flexibility allows the possibility to react or change with little penalty in time, effort, cost or performance.[115] In production or logistics, companies face new products to being able to be moved between two places for e.g. intermediate storage, further production or to ship. Therefore, TUs have to be added, removed or exchanged. Furthermore, Terkaj[116] defines four basic dimensions:

- Capacity: the execution of similar functionalities independent of system size.
- Functionality: different kinds of operations can be executed.
- Process: capable of operation diversity[117] to find one solution.
- Production planning: the system is able to change the order of operations to find the same result.

As described previously, capacity can refer to scalability in this work. Similar to the referenced intralogistics supplier, the other three dimensions have to be adapted to the domain in intralogistics routing. Functionality describes the operations one TU is able to execute, e.g. forwards or backwards transportation, or changing of the orientation

[114] This kind of transportation solution is a purely sequential transporting container. During execution, a specific state is provided before further execution or decision making. This ensures that the device only operates in specified, well-defined states. When an error occurs the device is in a proper and safe state.

[115] See [ElM08, p. 50].

[116] See ibid., p.52.

[117] By means of different kinds of operations in type and order.

of the container. Process refers to the possibility of finding different routes that can be compared to each other for selection. Finally, production planning is the reaction to changes during route execution, e.g. reroute to increase throughput, avoid blocking, deadlocks.

A flexible system has an additional advantage during run-time. The possibility of flexibility in the process and production planning helps to avoid issues by break-down or scheduled maintenance. If the break-down of a TU occurs, the method has to reroute containers by finding a detour to the destination based on the current position in the system. On the other hand, during scheduled maintenance, employees access the fully automatic system, which results in the shut-down of a transportation system or a partial part of the system. Still, the rest of the system should be functional to keep its availability[118] high.

Robustness

In transportation systems, units can go off-line for numerous reasons. On the field level most predictable one is a scheduled shut-down to access an automatically restricted area for restrictions vary regionally, e.g. Europe the ISO 12100[119] applies. On the other hand, unscheduled shut-downs can be caused by a failure of the machine itself in the sensor or motor and resulting in maintenance of a TU. For both, scheduled and unscheduled shutdowns, a part of the system is unavailable for an amount of time, or in a total shut-down.

At the PLC-level, the execution itself can cause a failure of a transportation system too, when routing is scheduled and the path distributed and scheduled at each intelligent device. In a fully decentralized system, a failure in synchronization can cause a deadlock when inverse flows occur at bidirectional transportation paths in the systems. This is when two movements from TU n_i to n_j at time τ_k are scheduled and executed, but at τ_{k+1} an inverse flow from n_j to n_i is also scheduled. When the first flow arrives later, e.g. due to blocking in its previous transportation, it interferes with the other flow, and a deadlock occurs because the first container for transportation at τ_k arrives at time τ_{k+1} when the other container has to be transported in the opposite direction. As a result of

[118] Availability of transportation systems is the derived quantity of the switched-on time and the down time, see [VDI04, p. 2].
[119] See [Sta10].

the deadlock, n_i and n_j are not available for any further transportation as long as the deadlock has not been resolved.

On the process-control level, another factor that can result in a non-usable part of the system is the material flow control itself. The minimum requirement for a routing method is to find a path between n_i and n_j when one exists. When parallel and bidirectional flows exists, a deadlock-free routing, time-slots[120] and detours additionally have to be considered. Furthermore, considering complex material flow systems, especially regarding the intralogistics supplier, huge networks with many TUs have to be considered as well. Given by the computational limitation of PLCs the routing method has to find a path in an acceptable amount of time[121] without global state knowledge.

In summary, a routing method is robust if the following applies:

- Proper reaction to partial system failure (field-level): If a path exists when a part of the system is down, it will be found.
- Reliable route finding: Planning of time-slots and flows for transportation, such that no deadlock can occur, in an acceptable amount of time.
- Synchronization of decentralized transportation units (PLC-level): During execution, the order of tasks planned locally at one intelligent device will prevent deadlocks.

2.3 Challenges for routing

Based on the previously described requirements in transportation in automated decentrally structured complex intralogistics material flows, the following section defines the challenges for a routing method to provide a transportation path for a container.

In summary, the routing method has to provide the following functionalities to operate: operational basis, evaluation of transportation units, path determination, path selection, and finally path execution. All five components are affected by the restrictions given by the introduced intralogistics material flows. Three main properties have to be considered,

[120] A time-slot is a defined period of time between τ_k and τ_{k+1}.

[121] Acceptable is when the container does not have to wait for route execution at a source s when arriving. A delay caused by the computation is not acceptable by any means because this would result in a blocking of s which could be a destination d for another good.

which are scalability, flexibility, and robustness. The interdependence between the functionalities and properties are given below.

2.3.1 Challenges for protocol structure for routing

The protocol structure needs to be the operational basis for the routing. Therefore the following properties have influences:

- Scalability: The structure has to have the ability to increase linearly when new TUs are being connected.

- Flexibility: Adding or removing a TU has to be done with a small effort such that other TUs are not blocked by any computation for too long.

- Robustness: When adding or removing a TU, the system has to be stable, and undefined states must be avoided.

2.3.2 Evaluation of transportation units

Since a routing is dependent on a cost model, this to developed model must have the following properties:

- Scalability: The model has to provide a linear complexity in computation such that the computational time can be estimated based on the size of the system.

- Flexibility: Adding and removing of TUs must be possible and being considered in the model.

- Robustness: The model has to reflect the transportation; otherwise, the computed values differ too much from the real times, and blocking or deadlock can occur.

2.3.3 Routing with local information

Since global information is missing, each TU must be able to collect and spread information through the system to provide proper transportation for containers. As previously noted, the routing itself consists of three parts. The first part is the path determination, where one or more paths should be found. Then, one path is selected, and subsequently executed.

- Scalability: The model for path determination and selection has to scale linearly to ensure a proper path for a container in an acceptable time.
- Flexibility: The routing should react to changes during run-time.
- Robustness: While routing multiple containers, a blocking has to be resolved.

The next chapter introduces methods that cover parts of the introduced attributes. Their advantages and drawbacks are also covered, such that the new method proposed in this work builds on the advantages while considering or resolving the drawbacks.

3 State of the art

So far, research has handled decentrally organized environments using a centralized unit for organization and control (see section 3.1.1) and fully decentralized control (see sec. 3.1.2). However, all proposed methods[122] lack the ability to handle fully decentralized environments with bidirectional flows simultaneously without explicitly transforming one bidirectional transportation unit into a unidirectional one to reduce complexity.

This chapter discusses current techniques to solve partially the problems described in section 2. First, section 3.1 introduces techniques in the same domain of decentrally controlled material flows. However, these lack the possibility to find routes and react to changes in an acceptable time for the highly complex system. Further, some lack the possibility to route, or even worse to handle inverse flows, so that they block flows until one routing is completely finished in the transportation system.

Afterwards, section 3.2 covers representation techniques used as an operational basis. Specifically, an overview of formalization is given, allowing an operational basis in transportation by routing.

Then, section 3.3 focus on the local evaluation of one transportation unit. It distinguishes by local metrics and end-to-end metrics.

Finally, section 3.4 discusses routing methods from other domains. The focus is in particular on the domain of high complex routing in networks, which is analogous to container transportation, with a few exceptions. One exception is that a data package can be deleted and/or duplicated. Therefore, such methods do not focus on a reliable and robust data transmission by invoking deletion or duplicating data packages. Similar to the previous section, a classification and evaluation of these methods are given.

[122] To the best knowledge and belief researched.

3.1 Research projects for decentrally controlled transportation

This section introduces current and past research projects regarding transporting containers in decentrally controlled transportation systems. It distinguishes between partial and fully decentrally control. The main concept relies on a control unit that has global or partial global information and computes decisions based on collected information in a master and slave manner.[123] In fully decentralized control, each entity[124] computes its own decision before execution. Therefore, they depend on collecting information in the system.

3.1.1 Partially decentralized control

Partially decentralized control focuses on few decision-makers which generally have higher computation power compared to other computation units (e.g. PLCs, industrial server). This results in a strict monolith hierarchy when each decision-maker‘s solution is used for the overall system strategy. In comparison, in a strict monolith hierarchy, each computation unit‘s decision making is independent of other units, and each solution is propagated to a computation unit with higher order.

MATVAR

In MATVAR[125] (German:"Materialflussysteme für variable Fertigungssegmente im dynamischen Produktionsumfeld; English translation:"Material flow system for variable production segments in the domain of dynamic production), developed at the Technical University of Munich, introduces a partially decentralized control. Its aim is to structure the segmentation or so-called islands in the domain of production and to define a variable material flow system considering practical situations. This is based on the structure requirements which has been determined for intralogistics physical and informational logistics to realize components for the material flow system.

[123] Master and slave manner based on the PLC concept. Master computes the decision and slave executes them.
[124] An entity is a material handling equipment with an processor unit, e.g. PLC.
[125] [GR00].

To realize the introduced working packages in [GR00], a prototype of MATVAR has been built, as shown in fig. 3.1.

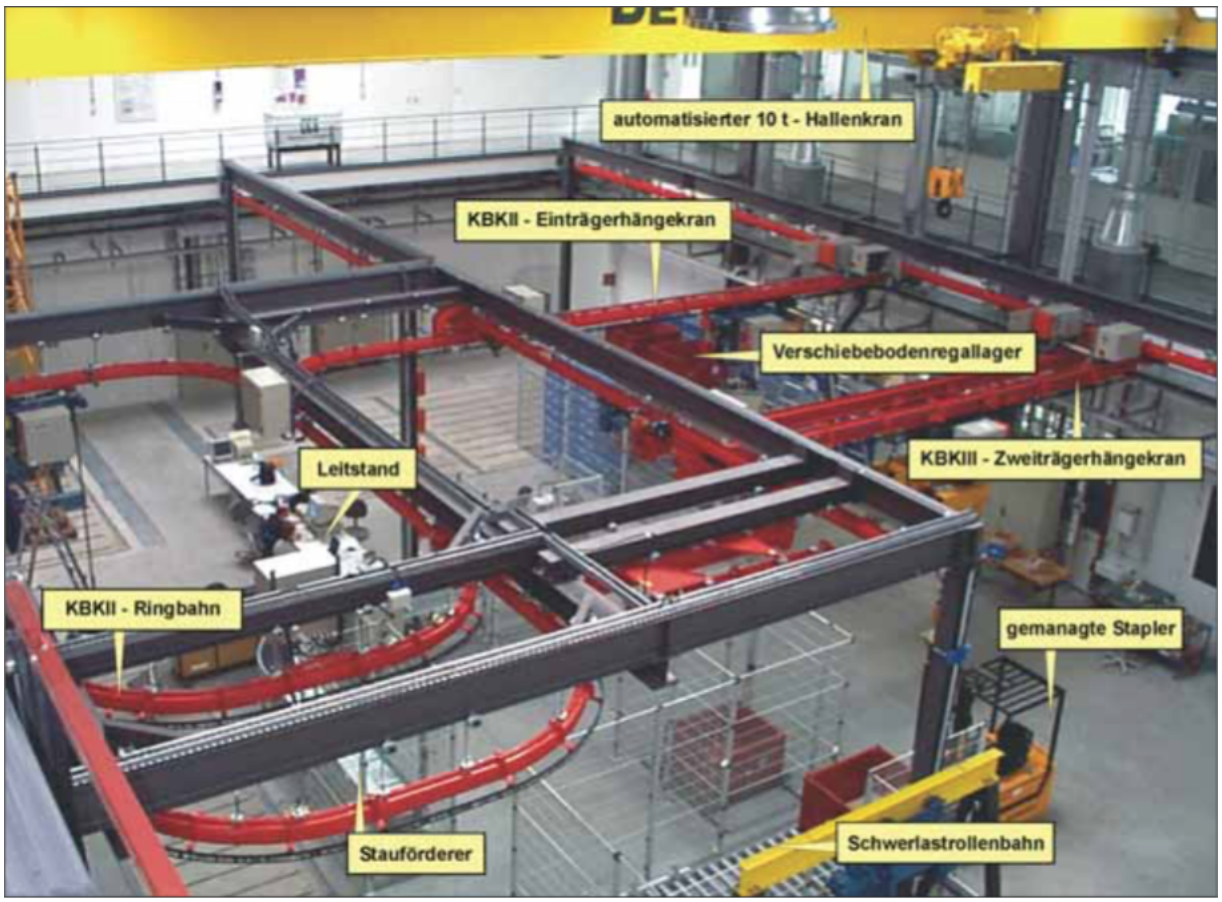


Figure 3.1: MATVAR lab at FML at the technical university of munich.

The prototype consists of a discontinuous conveyor for transporting units from storage, and between in and outfeed positions and productions units. The transportation system for finding and executing routes is defined in the working package 4. Its goal is to control in a decentralized way each transportation unit in the system and the organization of tasks in the system related to transportation.

The intralogistics material flow used at MATVAR discontinuously organizes and executes material flow computed at each PLC responsible for one cell. This is done by defining the logic for each cell at a low level of control, and an agent-based structure for the high-level control responsible for coordinating the communication between cells. Then, a local entity, a so-called environmental model server, computes each parameter for

local execution[126] by collecting the information representing the current system status. Therefore, a partial distribution organized by cells is used with a server unit to compute the execution steps for each cell. Thus, this method does not enable a fully decentralized control because the central server is needed. Furthermore, it lacks in scalability, and the bottleneck is the computation power of the server computing the single steps for each cell.

KARIS

Karis[127] is an autonomous vehicle capable of providing continuous and discontinuous transportation. By picking and driving from the source to the destination alone or in a cluster, different kinds of structural containers can be moved. Thereby, it acts like an autonomous guided vehicle. For higher throughput, the cluster can form a conveyor line for continuous movement. To enable operational transportation its structure is split into three layers mechanically and electronically. The lowest layer is responsible for moving on the floor based on one engine and four wheels; the second layer scans the environment for scouting and decision-making; and the third layer is equipped with a conveyor to move the container on top of the vehicle or for continuous transportation in a cluster.

To find a proper routing for transportation, two algorithms are implemented in competition. The first one is called Partial Build on Exploration (BonE), and the second one dRandom. BonE[128] operates by choosing one vehicle as a master element. This master-element computes the fastest route with the heuristic A*. Then, the master moves next to the source and records all obstacles on its way. Afterwards, the master calls a worker element which is at the farthermost visible position to the master element. It then drives next to the master and records all obstacles on its way as well. This calling and driving to the latest element is done repeatedly until the first element reaches the sink. If an obstacle occurs during planning, the master element recalculates the route from the source to the sink.

The second algorithm dRandom[129] is a decentralized method to establish a transportation between a source and one sink. Hereby, the last requested element requests the next element until a transportation line has been formed. If an obstacle blocks the way to

[126] Local in terms of each cell respectively.
[127] [Sto14].
[128] Ibid., p. 72.
[129] Ibid., p. 85.

form a line each element transmits its collected information about the environment back to the previous one in line. This is done until the first element is capable of redirecting the transportation line on a different path.

This method scales linearly in mechanical equipment regarding to the distance. Given by the size of the transportation unit as a square l^2 the number of elements needed for one route is minimum $\frac{d}{l}$ with d as the distance between the source and the sink.

There are several disadvantages to using one or both routing algorithms with fixed sized transportation units. First, is that one central unit (master-element) computes the route based on uncertainty in knowledge of the environment. Parallel, time windows, and fork or join operations are not considered in route finding. Therefore, several sources or sinks have to be handled during one time window, and a new transportation route has to be formed. Furthermore, it is highly unlikely that there exists no no gap between the transportation lines, which would result in breaking the continuous transportation with at least one discontinuous transportation of one element to fill that gap. Secondly, a transportation unit mounted on one autonomous transportation vehicle results in higher investment, operational, and maintenance costs compared to regular transportation units.

Multi-agent for transportation systems

A multi-agent approach for a real transportation system is evaluated in [LGM13]. The transportation system consists of several robots, machines for production, an automatic storage system, and 45 conveyors, 32 intersections and grippers for holding pallets. The main task of this system is delivering parts from the storage system to the manufacturing components. Each intersection is equipped with one controller managing the conveyors, intersections and grippers for transporting pallets. This method takes partial advantage of the decentralized configuration of the system in two layers. The low-level layer is decentrally organized and controls the input and output of the sensors and motors locally at each intersection. On top, the high-level layer is a centralized multi-agent method operating a transportation task on a routing table previously determined by the Dijkstra algorithm. The routing path is determined by the high-level layer and proposed for execution on the low-level layer. When a breakdown occurs, the low-level layer notifies the high-level layer‘s agents to reassign the direction of conveyors, such that

the underlying graph representing the transportation system is a connected graph.[130] This method lacks the dynamically use of bidirectional transportation of conveyors, such that a possible parallel transportation (n_i, n_j) from one node n_i to another node n_j would not be used, because a redirection is only be considered during breakdown but not congestion between two nodes caused by high traffic. Furthermore, the method takes no advantage of the decentralized control structure, and its scalability relies on the computation power of the system operating the high-level layer.

3.1.2 Fully decentralized control

In contrast to partially decentralized systems, fully decentralized controlled systems distribute tasks among intelligent devices for control or decision-making. Thus, no central unit observes, controls, or makes decisions regarding tasks during system run-time, and therefore these systems do not rely on global state information. This shifts the problem state from central decision-making, which results in a proportional need for computational power regarding the complexity given by the algorithm and the number of entities needed for decision-making, to a lower need for computational power distributed throughout the system.

Development of a completely decentralized control system for modular continuous conveyors

[May11] introduces a decentralized routing for continuous conveyors. It splits each component into modules and defines a specific task for a period of time. Tasks are unidirectional transport or redirection of a container at one module based on a predefined route.

To determine routes, each module stores a matrix representing the complete transportation system. This matrix is determined using the distance vector algorithm[131] by first creating a matrix with all reachable targets and the total costs by using a determined successor in each step, respectively. Afterwards, this local matrix is shared with the neighbors and if a path to a destination by a different successor exists, the matrix is

[130] Such that each node can reach another node by a path.

[131] Distance vector algorithm operates in decentralized controlled system and is based on the Bellmann-Ford-Algorithm to determine the shortest path from a source, see [MD01, pp. 14-23].

updated at each successor module. This initial local matrix computation ensures system operation during run-time by providing the ability of route finding for one container at arrival.

In operation mode, during the arrival of one container at one module, a route reservation algorithm starts. Based on the local matrix, the route from the current module (source) to a predefined destination is selected. Afterwards, each module on the path from the source to the destination is triggered for route reservation. If all modules accept the reservation, the route is blocked for this container and the transportation can start. If at least one module rejects the reservation because a different container is reserved in the opposite direction, the source module computes an alternative route based on the local matrix.

Because the local matrix is fixed after the initial phase, each container is transported on the shortest path. This results in an overload of transportation at this one specific path. A parallel path p_2 to a shortest path p_1 with costs $\text{cost}(p_1) = \text{cost}(p_2)$ is never be used for path stabilization and transportation path splitting. This is due to the deterministic behavior of path selection in the local matrix. The first shortest path is always selected.

Secondly, due to the behavior of continuous transportation, instead of time-discrete transportation, an occurrent transportation of containers with a path p_1 for container c_1 and p_2 for c_2 with an inverse flow given in p_1 and p_2 is not considered. This results in a blocked path p_1 for c_2, and for c_2 a path with higher costs is chosen. Otherwise, if no path p_2 exists, the transportation is blocked until p_1 is available.

Thirdly, considering the control hardware described in section 2.1, a dynamic non-deterministic array has to be used to store the local matrix in a general manner (so that all possible systems can be represented). Therefore, specialized hardware enabling such an array must be used, which would not conform to [DIN05] [132].

[132] Considering a regular transportation unit with a specialized hardware fulfilling the regulation to provide the necessity of commissioning will costs around $\frac{1}{4}$ of the mechanical costs of the transportation unit.

GridStore

The GridStore[133] method is a rule-based decentralized algorithm to find and execute a route for a container. It basis operates on a chess-based structure and is split into two methods: one for north-south and one for west-east movement.

The first method finds an orientation and alignment in a north-south manner, and the second one in a west-east manner. To this end, each element in the grid uses its own status and that of its neighbors. The routing itself is done by defining a home grid such that a container enters one element in the north and needs to be moved to one element in the south. First, the north-south manner elements send a request for moving to the element holding the container close by. If the element‘s status allows a movement, the container is moved in a tandem manner by sending a commit message. If the destination row is different to the source row, the west-east method checks whether an element can be moved in a similar way as the north-south manner, but column-wise.

This method prompts the routing by executing directly when possible. Therefore, no global or neither local optimization is done, since each element will transport the container when possible. As a result, containers with inverse flows are not rerouted beforehand and can, therefore collide at adjacent elements. They have to be moved in a costly zig-zag manner to avoid this.

Plug and produce policies for reconfigurable transport systems

[CCOR14] Introduces an approach that follows the bidder and auctioneer pattern.[134] After determining the connectivity graph, each node n operates as a bidder and one as the auctioneer. The bidder n with the highest utilization factor wins the auction and is allowed to execute its transportation task. Each bidder‘s utilization factor is computed based on the distance to the destination, a conflict detection flag and local cost value. The distance to the destination is a straightforward factor considering the geological distance between the node n to the destination d. A conflict detection flag represents the possibility of a conflict and results in a deadlock. It is a binary flag checking for a current existing inverse flow, and additionally whether a container has visited a node n more than once without any further processing. The last value is the local cost value

[133] [GFSU14].
[134] [KTL+06, pp. 1625–1629].

representing the local costs of accepting the container in terms of energy and time. When a container arrives at a node n_i, an auction begins and each node n_j in the system bids. It is differentiated between a group and a single bid. A single bid is a bid done by only one node n_j whereas a group bid is a summarization of single bids in a previously formed group of nodes.

Similar to the previously introduced methods, this approach does not take into account time windows and therefore does not maximize flows when parallel flows would be possible. Instead, a binary flag, the conflict detection flag, sets a path to a specific direction only. Hence, two containers cannot move in inverse directions on one path even if the time difference is large enough to make it possible without a deadlock. Furthermore, this method would cause a network collapse in systems with a high number of nodes and containers being simultaneously handled. Each node n_i that becomes an auctioneer floods the system for a bidding request in the system. Finally, each node n_j in the system responds to this request, which causes additional network traffic and the necessity of computational power to handle each request.

3.2 Protocol structures for routing

All previously introduced methods lack the ability to represent the system status and its changes over time. Therefore, they are not able to dynamically determine routes based on occurrences like blocking, break-downs, and inverse flows. This leads to the need for an operational basis able to represent changes over time and allow a prediction of future states to handle such occurrences.

A wide variety of well-known representation bases exists, e.g. regular graph,[135] Petri-Nets,[136] UML,[137] Z,[138] etc. as well as more specific mathematically models that enable the fulfillment of specific operations.

With a formalization basis, as stated in [WIIA12], main concerns are efficiency/performance and reliability, whereas security and scale-ability are rarely considered. A formal basis, compared to an informal one, provides the ability for prior system analysis and

[135] [Wal10].
[136] [SV88].
[137] [SKH07].
[138] [Spi92].

evaluation. Therefore, a formal basis provides system correctness by mathematical proof, thus enabling correctness and eliminating ambiguity, as stated in [Hog02, p. 3].

3.2.1 Transportation system representation

First, the following provides an overview of existing formal methods which enable a modeling of a system for later operational. [BCDW04] divides formal specifications, which provide a basis, into four categories:

- Graph - Graph grammar provides the capability of graph representation, and the graph itself the system architecture.
- Process algebra - This kind of formalization is specialized to analyze concurrency in a system.
- Logic - This is a formal basis for dynamic software architecture specification, e.g. Z,[139] VDM,[140] PVS,[141] to define a systems architecture and its behavior for later analysis.
- Other - Formalization which do not fit in the three previous categories.

It can be concluded that the well-studied graph grammars are advanced in reconfiguration, operational, and still enable a high expressiveness and scalability. Each different class within the categories is specifically designed for some tasks[142]

3.2.2 Graph representation

Graphs are widely researched ([Wil96], [Wal10], [Büs10], [Ruo13] or [GR14]), accepted, and used as a system representation for routing. In short, a graph G is a tuple $G = (V, E)$ with a non-empty finite set V of vertices or nodes[143] and E as a set of edges. An edge e_i is a tuple $e_i = \{v_j, v_k\}$ connecting two vertices v_j and v_k. A walk w_i in G is a sequence $w_i = \{v_1, v_2, \ldots, v_i\}$, if $\forall v_i \in w_i \nexists v_j \in w_i : v_i = v_j$ is true the walk is called a trail, and otherwise a path. If only $v_0 = v_i$ is fulfilled, w_i is called a cycle.

[139] [Spi92].
[140] [RAA11].
[141] [AR02].
[142] For a more detailed explanation see .([BCDW04]).
[143] Both definitions are interchangeable, such there exists no difference as stated in ([Wil96, p. 8]).

Regular graphs (or static graphs) are not capable of capturing the dynamic behavior of transportation systems, as stated in section 2.1, such as vertices not connectable to a function and dynamically changeable edges weights representing the current system connection status. Such extensions to the regular graph provide additional functionalities as listed in [HSS15] and [BBD16]. [BBD16] classifies graphs as follows:

- (un) directed - as stated above.
- weighted - adding $E \times \mathbb{R}^+$
- compound E_i^T forming a hierarchical tree
- multivariate $G = (V, E, W)$ with E as weighted edges and W as a function retrieving the weight for each $v \in V$.
- dynamic $\Gamma := (G_1, G_2, \ldots, G_n)$ with G_i as static graphs and i referring to time steps t.

Each element within the classification can be combined such that e.g. a weighted multivariate dynamic directed graph, is still a valid and operable graph. In particular, dynamic graphs are able to handle and provide the possibility of changes over time. These changes could reflect weights and removable and/or addable connections, which represent the previously occurrences.

3.2.3 Network structures

The evolution of routing methods has been mainly driven by the network field in IT.[144] Methods like distance vector routing using the Bellman-Ford algorithm and link state routing based on Dijkstra represented first attempts at routing. Subsequently, hierarchical routing was introduced to meet the need for a network structure to represent the Internet. Because of their flexibility, low cost, and ease, wireless networks became dominant compared to wired networks, e.g. Wifi-based WLANs, ad-hoc networks. The following considers ad-hoc networks by comparing the needs described in section 2.1, like connectivity and especially the need for dynamic handling of project specific network structures. Ad-hoc networks enable communication but lack a fixed network structure.[145] To overcome this lack, new research fields have been created, such as

[144] [Cha15, p. 2].
[145] [KM07, pp. 324–339].

network configuration, device discovery, topology maintenance, ad-hoc addressing, and self-routing.[146]

3.3 Evaluation of transportation units

After defining an operational basis[147], as described in the previous section, each node in the graph needs to be evaluated to enable a comparison. In a later stage, this enables a definition of a cost model such that a path can be evaluated and compared to other determined paths if they exist.

3.3.1 Local metrics

The main task of local metrics is too evaluate one node during path finding. Several approaches exist, by focusing on one node i only and its local characteristics, which could be the distance to another node j, its local costs for transportation and estimating costs for future transportation. This section investigates all three possibilities for evaluating one node i.

Local Routing Metrics

In [ZR03a], the distance from a current investigated node to the destination is evaluated. The result of path finding, when only considering the geological position, is the shortest path in terms of physical distance of the node to the destination. Routes in between are unobserved. The transportation-specific variables, e.g. transportation time, disturbances, are not considered. Thus, a short route is selected in terms of geological distance of the node to the destination, but not the fastest one. Especially in a highly dense transportation network, the overload of one path is neglected, and future transportation tasks are executed on this specific overloaded path.

A simple approach is given in [YYWL05]: a mesh is made of possible paths, prior system run-time, and stored at each node. The mesh stores the hop count to each reachable node and link weight characterizing the link transmission speed. To establish a route, each

[146] [WS04, pp. 29–31].
[147] In this case the dynamic graphs.

next hop is chosen opportunistically with the highest link weight possible in reaching the destination. Paths with a lower initial link weight are not considered during route finding. Therefore, if a faster route starts with a lower link weight, it is not considered because the concurrent link with a higher link weight is preferred.

Metrics based on expected distance to destination

This category of metrics describes methods using an approximation for packages transmission. In [DC12], each link between nodes is given a probability of reaching the destination. By combining the geographical position of the node and the evaluation of each link between nodes, a more robust routing can be achieved. This is given by considering the link with the highest probability only. Analogous to container transportation, the probability reflects the robustness of one link in terms of downtime. Similar to the previous method, this one does not consider the overload of a specific path for one route.

Expecting one-hop throughput

As an advancement to the two previous methods,[ZLYB07a] introduces link evaluation between two nodes. Besides the evaluation of the node in terms of geological distance to the destination and the probability of transmission, the delay in local transportation is also considered. This is given by, e.g. the transportation time on one node itself, and preparation to enable a transportation. Only a static local delay is considered, hence a delay due to environmental changes is not depicted.

3.3.2 End-to-End Metrics

In contrast to local metrics, end-to-end metrics evaluated the route between the node and the desired destination of the container. A simple method is OPRAH,[148] whereby the network is flooded by broadcasting a route request message RREQ from the source s until the destination d has been reached. Each node n_i in the network, when it receives the message RREQ, increases the hop count $h_s + 1$ with $h_s = 0$ beginning at s, creating a duplicate message m_{i+1}, and broadcasts m_{i+1} to its neighbors. Every receiver n_j of

[148] [Wes06, pp. 570–573].

a duplicated message stores this information in his local routing table with the hop count h_s from the source s to itself n_j. Because of the nature of broadcasting and its resulting flooding, a duplicated message RREQ_{i+n} is highly likely to arrive at a node n_i if a connection exists. If several connections exist, several messages with different paths from s to the current node n_i may arrive. Then, the old hop count h_s from s is compared with the alternative path's hop count h'_s. If $h'_s > h_s$ then the routing table is updated and the costlier path is dropped[149] from the routing table. When a message arrives at destination d similar steps are executed. A route reply message RREP is created and broadcast until received by s. Each node n receiving RREP updates its route table and marks the route as valid. If $h_s > \gamma$, where γ is a predefined threshold, is true, then RREP is dropped to reduce the maximum length of a path and infinite loops in the system. When RREP arrives at s the message can be sent in the sense of RREP. Thus it is broadcast through the network.

Similar to the previously introduced work by [LZS08] adaptions have been made to reduce the number of messages being created and forwarded to reduce the length of the route itself. In addition, the closest neighbor to the destination is chosen. Instead of only monotonously counting the hops, this method measures the ping[150] at one link n to n_{+1}.

This methods rely on broadcasting and therefore flooding the network with messages. A network with a high density of sent data would collapse due to the nature of broadcasting. Instead of taking advantage of the generated routing tables and forwarding the data in a message directly based on the knowledge stored in the routing tables, the data is broadcast as well.

This method operates as follows: The first step is storing a routing table with h_n which is a simple table with all reachable nodes n_m. In addition, γ is network specific when $\gamma < h_{sd}$ with h_{sd} as the minimal hop count from s to d no message would arrive because RREP would be dropped beforehand. A high γ would cause a high flooding of the network because messages would roam in the network before being dropped.

[ZR03b] uses a similar approach to [Wes06] and the same contention-based routing approach as [LZS08], a geographical analytic analysis firstly in terms of an average

[149] Dropped in the sense of being removed from the system and not further distributed in the network.
[150] Measured time between sending and receiving a message.

number of hops to the destination. This is done by defining a lower and upper bound of hops which is characterized as a statistical evaluation.

In addition to taking the ping of a link into account, in [ZLYB07b] the time to the destination is measured as well. Therefore, to determine the route a metric is used weighting the distance and travel time between nodes s and d.

MGOR[151] extends the metric used in [LZS08] by considering the transmission time next to the delay at one link between two nodes n and n_{+1}. Furthermore, it uses a heuristic to determine the route to d. The node selection is done by computing the current state of n and time to the destination d by the mentioned metric.

3.4 Routing with local information

Routing has been widely researched, and insight on this topic is therefore adapted to container transportation in complex dynamic environments such as an automated decentrally structured highly complex material flow with thousands of conveyors. In [May11], a differentiation between network routing and routing in material flow is given, as described in section 3.1.2.

In table 3.1, [May11] compares regular data transmission with material flow. This table has been extended to provide a current view of state of the art. Similar packet splitting approaches exist in material flow e.g. build-up and break-down of existing containers in warehouses integrated into shift planning,[152] and order picking in automatic warehouse-systems.[153] Moreover, especially in air cargo terminals, these operations are a standard solution for packing ULDs.[154] Speed is limited mostly if people are interacting with machines during runtime: then, it is limited to $0.1\frac{m}{s}$ and further safety restrictions apply, as stated in [MK11]. Furthermore, if two packages collide, a deadlock is not conclusively given in network routing. Current routing methods with a detour ability reacting online are able to remove the edge in a network graph G_n such that a new graph G_{n+1} is created and plan new routes for both deadlock-causing containers. Such methods are introduced in section 3.4.1. Buffering nodes during routing are comparable

[151] [ZYL09].
[152] [RG09, pp. 725–739].
[153] [AGA99, pp. 501–515].
[154] [VT93, pp. 159–166].

Attribute	Data transmission	Material flow	State of the art
Packet splitting	possible	not possible	build-up & break-down
Speed	approx. 300.000.000 $\frac{m}{s}$	approx. $1\frac{m}{s}$	0.1-5$\frac{m}{s}$
Consequence of a collision	low: resend the data	high: system is blocked	detour has to be planned
Simultaneous path occupation	not possible	possible in the same transport direction	
Buffering in nodes	possible	not possible	short-time buffering possible
Focus of optimization	utilization of the ether	minimization of path distance	project-specific utilization.
Network size	internet approx. half a billion	usually less than 100 nodes	several thousands

Table 3.1: Extended comparison of electronic data transmission and material flow based on [May11].

in routing to just-in-time delivering as explained in [Wat95] and an example is given in [OKMS16]. Similar to an edge representation[155] that could shortly store containers as evaluated in network routing in [PKP13]. Such methods are widely used for securing the network against attacks.[156] The network size depends on the domain in which the routing operates. Small networks exist for small warehouses, semi-automated operating warehouses, or terminals. In contrast, large, fully automated terminals exist, e.g. British Airways World Cargo Centre, which is fully automated or as a further example the system for evaluation with several thousand intersections and conveyors as explained in sec. 7.1 given by an industrial intralogistics supplier.

3.4.1 Opportunistic Routing

All previous introduced routing protocols rely on global state knowledge. Whereas, its provided by a central unit, e.g. a node n with "god-like"[157] knowledge and behavior. Transferring the routing paradigm to container transportation, a network overflow would

[155] E.g. bidirectional conveyors connecting intersections.
[156] [CAG15, pp. 1807–1812].
[157] In the sense of global state knowledge of every n and instantaneous state updates of each n during run-time.

cause a container overflow at specific nodes n. Thus, robust methods must be found that can firstly decentralize this "god-like" knowledge, such that it becomes possible to receive status updates from a specific number of nodes n for decision making, for route finding. Secondly, these methods should allow load balancing during run-time to overcome unforeseen situations, e.g. blocking. Thirdly, these methods should be able to determine the failure of a node n in transportation such that in real-time a detour can be computed and being executed.

N. Chakchouk's survey[158] provides an extensive historical background of routing paradigms. Highly complex material flows in industrial environments are characterized by their higher degree of requirements in safety and therefore reliability and robustness compared to enterprise networks as stated in section 2.1. Still, dynamic and decentralized routing is needed to enable scale-ability and therefore a reduction in cost in software and electrical construction, installation and commissioning of mechanical equipment. Therefore, according to Chakchouk,[159] the opportunistic routing advantage lies in reduced topology information and maintenance. In addition, benefits compared to traditional routing paradigms are flexibility and easy adaption to topology changes, since a node n itself stores only partial global state information to enable at least a forwarding to a successor capable of reaching the destination by forwarding or direct access. Therefore, opportunistic routing is the ideal candidate for a fully decentralized routing method capable of finding routes for containers and ensuring deadlock-free transportation between the source and the destination. The following sections present an overview of the current state of the art in different categories of opportunistic routing.

Geographical routing

These sections review current work in opportunistic routing mainly relying on geographical information for routing. Therefore, their optimization goal is to minimize the traveled distance, which authors mostly assume to be strongly related to transportation time. Especially in container transportation in industrial environments, the transportation distance and transportation time are highly coherent. This is because a physically element is bounded to the machinery transportation specifications.

[158] [Cha15, pp. 2214–2241].
[159] Ibid., p. 2216.

A robust approach to opportunistic routing is done in ROMER,[160] explained in section 3.3.1. Based on this, further methods have been developed, such as SRSNR,[161] BLR[162] and DTRP.[163] Both, BLR and DTRP methods relay on sending duplicated packets (multihop) to increase the reliability of message parsing in the network. SRSNR uses so-called beacon nodes to determine the route from s to d. Each beacon node holds the traveled path from s to d as route r_{sd}, and h_{dn} as the hop count from d to a connected beacon node n_i. Thus, connected beacon nodes receive periodical updates by each d with current hop counts h_{dn}. The update period time is parameterizable and therefore tunable to the specific network. Each n_i stores the information inside the beacon node for route finding. Therefore, based on h_{dn}, each n_i knows the distance to d and the path r_{sd} by determining the subpath $r_{nd} \in r_{sd}$. Similar to OPRAH, only h_{sd} is considered, and not the link status or performance. Thus it is assumed to be homogeneous through every node in the network.

BLR is one of the first methods to use the current position of n_i to compute and execute r_{sd}. It uses three modes to reach d. The first mode is a so-called greedy-mode, which determines the neighborhood of n_i within a diameter. In contrast to the previously introduced methods, BLR uses the real position[164] of n_i instead of a fictive position represented by a hop count. The second mode is a backup mode, which is triggered when no neighbor is reachable within the diameter. Then, the next reachable nodes n_i will receive the duplicated data package. The most effective way is to execute the transmission in the third mode, the unicast-mode, where the closest neighbor to the destination is used. This mode is repeated at each n_{i+1} until d has been reached.

DTRP is based on BLR, and each n_i is aware of its real position as well. Instead of only taking advantage of that position, a metric is used to evaluated n_i and the link between n_i and n_j. Based on this information, the next node n_j is selected and receives the data packages. The link is evaluated by a metric that takes into account radio irregaluation. Each reachable node n_j from n_i is evaluated by its distance in terms of real position. In addition, the reliability of n_j with regard to forwarding the package is evaluated by considering its available power and possible energy consumption for further data forwarding. The last element in the metric is to evaluate the movement of one node n_i

[160] [YYWL05].
[161] [NJE+07, pp. 670–678].
[162] [BHR10, pp. 96–107].
[163] [ZRBC14, pp. 2138–2143].
[164] Real position in means of world position e.g. determined with a GPS-like device.

in real positioning. It is assumed that a node n_i can be moved through the environment, e.g. a mobile phone.

A drawback of this methods is that it does not consider load balancing. Therefore, a specific route r_{sd} with a high current load of data transmission is used extensively until, in the worst case, a collapse occurs.

Link state aware routing

Compared to geographical routing, where the focus is on fast transmission between the source s and the destination d based on geographical position, link state aware routing considers and evaluates the links from one node n_i to choose the most sophisticated[165] neighbor n_j. Due to the nature of opportunistic routing in determining sub-paths between s and d during run-time, the current status of n_i is considered to be more up to date compared to conventional routing in determining a route before routing. Therefore, the method of evaluating the link between n_i and is neighbors is able to react to status changes.

One of the first link state aware routing methods is ExOR.[166] This method operates and makes its decision based on a matrix reflecting the approximation of the loss rate between a pair[167] $e = (n_i, n_j)$ of nodes. The matrix itself is built by broadcasting the current state of a pair e through the network. Afterwards, the candidate set[168] is determined by the hop count beginning with s and d. If several routes to d have the same hop count, the route with the lowest loss rate is chosen. This method‘s disadvantages mainly occur during the forwarder set[169] selection. First, this method relies on flooding the network to determine the candidate set that occupies the network; and second, only received link states received prior to the system start for a pair e are used without triggering a state update, meaning that the route is selected based on an outdated status.

A different approach is introduced in MORE:[170] it divides the network into segments; each fulfilling communication tasks, e.g. data transmission and receiving. As a basis, each node n_i in the network is considered to be a fully functionally PC enabling the transfer

[165] sophistication in means of e.g. reliability, transportation speed.
[166] [BM04, pp. 69–74].
[167] A pair are two directly connected nodes n_i and n_j. In graph domain called edges.
[168] Set of nodes capable of reaching the destination d.
[169] Set of neighbors at n_i capable of reaching d.
[170] [CJKK07, pp. 169–180].

of small, medium, and large sizes of data as well as specific control data. This control data is used to keep the network structure updated at each n_i. Hereby, each package is divided into chunks and later merged together at the destination d. However, MORE only considers coding[171] and transmitting separately, which increases the overall network communication. Therefore, CodeOR[172] introduces the ability to transfer windows and to sending and receive multiple chunks. In SlideOR,[173] this is improved by increasing the number of chunks being encoded, sent, and decoded online by introducing sliding windows. Such windows enable the handling of different chunks with different requests to be handled during run-time. A further improvement is introduced in [KWH10] by using overhearing and reduced overhead in sending duplicated messages. This is done using an acknowledgment structure in which the node first synchronizes itself with its neighbor before sending and receiving messages.

All introduced methods compute the link state status during pre-processing once and use this static value during route determination. Therefore, all lack the possibility of status changes during run-time. To overcome this structural issue, O3[174] has been introduced. O3 splits the network into two layers, called the overlay and underlying network. The underlying network is the physical representation of the network itself. Thus, one node n_i in the underlying network is exactly one physical device capable of message handling. Based on this underlying network, the overlay network; it is a subset of the underlying network. Each n_i in the overlay network represents one n_i in the underlying network, but they do not have to be connected physically. Therefore, the underlying network optimizes itself on the physical level, and the overlay network independently computes the route of the current physical structure. Thus, both network operations are split and independently optimized.

Load balancing

Besides route finding based on the local state of nodes, the current flow situation must also be evaluated as well to ensure a robust routing, thereby preventing blocking of containers and, as the worst case scenario, deadlocks in the system. Congestion-aware opportunistic

[171] Here in the sense of splitting the data package into chunks.
[172] [LLL08, pp. 13–22].
[173] [LLL10].
[174] [HBQR11].

routing utilizes the flows based on information at n_i and links between n_i and it neighbors n_j. This utilization is used in route selection and subsequent execution.

The previously mentioned methods lack in load balancing. Each is capable of finding a route r_{sd}, but when it comes to mass transportation, they do not consider the advantage of parallel paths. Wherever a parallel path takes longer for transportation but harmonizes in terms of overall transportation time. Therefore, the following section focuses on detouring a container and considering parallel paths if existing.

Reactive balancing

One of the first congestion-aware opportunistic routing methods is ORCD.[175] It is based on a network model with time-slots $t \in \{0, 1, 2, \dots n\}$ for a time interval $[t, t+1]$. During each time-slot t the routing decision is made in a three-step manner: first, transmitting one data package to check whether a neighbor node n_j of n_i is alive; second, sending an acknowledgment message to show that node n_j is alive; and third, transmitting the data itself. All three steps are done once during a time interval t. The nodes replaying to the acknowledgment message are sorted by a cost method utilizing distance to the destination d, transmission delay, queue backlog[176] and optimal throughput. The drawbacks of this method are relying on a centralized scheduling algorithm managing the time-slots t for computing the cost methods for each n_i.

E-WLBR[177] was introduced to overcome the load balancing issue in networks with several paths. Using an energy-cost-model, each node‘s energy is estimated online, along with its link cost in the distance to d. When a specific node n_i is extensively used, its energy value is being reduced such that when a threshold λ is reached, a parallel path is used instead. Besides the energy value, the threshold also considers next to the energy value the distance too, to select the next shortest path to d. To enable the routing and determine its connectible neighbors, a routing table for each n_i is created by broadcasting and flooding the network from each d until s receives the flooding messages and drops it. This first approach is reactive instead of proactive. It reacts to the diminishing energy during transmission instead of evaluating it first, and thus reacts to this possible critical future situation.

[175] [NJ10].
[176] If a transmission request arrives within a time-slot t it will be postpone to $t+1$
[177] [BU12, pp. 31–37].

Proactive balancing

A decentralized approach is the enhanced congestion aware adaptive routing protocol.[178] Each node n_i performs a self-evaluation first before publicizing its availability and buffer for further data transmission. Self-evaluation tries to predict future flows and connections with other nodes, thus estimating a willingness factor for receiving messages from neighbors has been estimated. This factor is broadcast to each neighbor n_j of n_i. To avoid congestion because several neighbors select n_i for transmission simultaneously, a safety margin has been introduced based on the same factor, e.g. 10% from the buffer. The advantage of this routing protocol lies in the routing of n_i with a small buffer; by increasing buffer size, the congestion-aware method is not useful due to overloading the network by broadcasting. One disadvantage is the reactive evaluation of each n_i and high dependency of the future prediction method which has not been introduced yet.

In ORW,[179] a pro-active load balancing is done by duty cycles. To ensure a power safe transmission, each device is intended to sleep for a specific period, wake up to communicate with its neighbor, and then sleep again. ORW's main focus is energy efficiency and reliability instead of network throughput. The load balancing is done by sending the data packages to the first node waking up and receiving the package ready for transmission. They introduce EDC for the wake-up and sleep procedure which is based on ETX[180] and its successor ECTX[181] by estimating the expected time to reach a potential forwarder, the estimated time to travel to the destination and a constant value for forwarding the message to the next node. If there are more reachable neighbors, the sleeping time is shortened to increase the reachability and therefore the throughput. Load balancing is done by sending the first reachable neighbor the data package. A disadvantage is that there is no direct control of the load balancing: instead, the first reachable node will receive the data package.

Based on ORW, the ORR[182] method has been developed. Instead of randomly sending messages to neighbors, it manages a forwarder set of neighbors for package forwarding only. The forwarder set is controlled by only allowing a maximal number of neighbors n_{max} to decrease the number of multiple messages in the network and the resulting flooding of one specific message. Afterwards, each potential forwarder is evaluated based

[178] [KR14].
[179] [LGDJ12, pp. 185–196].
[180] [DABM03, pp. 419–434].
[181] [SGZJ12, pp. 1–9].
[182] [SB16, pp. 1–16].

on average waiting time for sending, the average number of packet transmissions and energy consumption to the destination d by estimating the future path to d. By using these mentioned estimations, ORR has a longer lifetime by load balancing the traffic within its neighbor set. A disadvantage is that ORR does not directly forward a message: instead, it multi-casts a message within its neighbor set and relies on duplicate messages to reach d which is not possible in container routing because a container cannot be duplicated.

A different approach is taken in DLBR,[183] SALR,[184] CAM,[185] and RBOLSR,[186] which proactively evaluate nodes and determines online detours. DLBR uses the shortest path technique to compute the path with the maximum throughput. This is done by using a metric to select high-quality links with lower congestion. The metric for one flow $C_F(n_i, n_j)$ between node n_i and n_j is $C_F(n_i, n_j)) = C(n_i, n_j) - f(n_i, n_j)$ with the link capacity $C(n_i, n_j)$ and flow $f(n_i, n_j)$. Afterwards, to determine the route, the minimum end-to-end delay (EED) is considered and second $C_F(n_i, n_j)$. If a tie exists between two routes, the route with the lowest hop count is selected.

The secure adaptive load-balancing route protocol (SALR[187]) consists of three parts, namely: adaptive load balancing, security based on node strength, route Prediction based on previously chosen routes. During congestion detection, each node n_i is being observed to estimate future congestion. If congestion is being predicted the dynamic load balancing of SALR takes action to prevent n_i from switching to congestion mode. This is done by recording its capacity C_n and going from an availability state to tending towards a congestion state. If n_i goes into the second state due to a high network traffic, it notifies all its neighbors about its state change. This triggers the determination of the node strength of the neighbors of n_i to compute a detour over a node n_j that is capable of handling the high traffic of n_i. To predict that a node n_i will go towards a congestion state, training data is collected during run-time. Therefore, each route is assigned a weight, which enables a comparison. During route determination, the weight of one route is taken into account and selected for network routing.

[183] [HBD15, pp. 450–454].
[184] [LSS15, pp. 1–5].
[185] [AP16, pp. 1–8].
[186] [YNA16, pp. 102–107].
[187] [LSS15, pp. 1–5].

In CAM,[188] similar to SALR, each node n_i is observed by a central party.[189] This central party evaluates the current status of n_i which contributes to a utilization function $u(n_i)$ for route selection. The utilization function to determine the route considers the distance from a subpath within the route instead of links only like in SALR. Furthermore, it evaluates the competition and therefore the current data load at one node n_i. This is done by defining the congestion level of n_i and its relative success rate of receiving a message. In addition, the buffer occupancy for n_i is computed and used for path finding by the utility function. Disadvantageously, this method uses a push-based protocol[190] periodically notify all nodes of its current status. This message broadcasting leads to overhead in message parsing. Furthermore, this method prioritizes a message m_i and drops m_i if its priority is lower than a message m_j and the buffer of n_i is depleted.

The final method introduced is RBOLSR[191] it is based on OLSR.[192] OLSR uses a routing table previously generated based on the Dijkstra-algorithm.[193] During run-time, each node n_i periodically broadcasts transmission control messages to update the routing tables of all other nodes in the network. Routes for traffic are categorized into "MAIN" route and "BG" route. The "MAIN" route defines the path which will be used for data transmission between a source s and a destination d in the network. The "BG" route is used for background messages, e.g. route table updating. RBOLSR uses a binary method to define whether the load on a node n_i is high or not. Therefore, prior to network run-time, a threshold δ is defined and determines a flag if a node n_i is currently facing a high load. To properly compute the route, the control message and routing table which are based on OSLR are extended by another field representing the flag. During run-time, this flag is locally computed by n_i itself and broadcast in the network. A disadvantage is that a binary flag does not represent the behavior over time; thus if the load of node n_i drops below δ it will be considered for future routing without having the ability to recover. Therefore, it is considered again for routing, which could trigger shortly δ, which could, in turn, result in another broadcasting of the current status of n_i as well, thereby also leading to increased traffic.

[188] [AP16, pp. 1–8].
[189] e.g. Server, Computer with global state knowledge.
[190] In a push-based protocol, the information source notifies all subscribers (n_i interested in this information) automatically and independently if this information is needed. In contrast a pull-based protocol node n_i will proactively ask a node n_j for new information; see [SCC07].
[191] [YNA16, pp. 102–107].
[192] [JMC01, pp. 62–68].
[193] [Dij59, pp. 269–271].

Each introduced reactive and proactive load balancing method is capable of route finding. In general, the proactive load balancing methods outperform the reactive ones. This is given due to proactively controlling the routes during the system run-time: while reactive methods react by rerouting when an overload situation is about to happen, proactive methods try to avoid this situation by evaluating nodes or links during run-time, and determining a detour, and rerouting when needed prior before the overload occurs, or is imminent.

4 Required actions

To develop a scalable, flexible, and robust method capable of finding and to execute transportation in decentrally structured highly complex material flow, several modules have to be provided.

First, a flexible elementary basis must be developed that is capable of reflecting the system graph respectively to its current status in time. Dynamic graphs have been shown to be able to alternate their structure in time such that nodes and edges can be changed, added, or removed. In addition, changeable weights and directed edges also have to be introduced as well to represent the transportation task, and especially to block a transportation direction on bi-directional conveyors to prevent deadlocks with the allowance of inverse flows.

Despite the currently used routing methods in material flows in decentrally organized transportation, the new introduced method must take advantage of the decentralized organization of transportation units and its intelligent devices to enable scalability and fast reaction due to graph changes online. This is archived by three main modules.

The first module is a flexible and scalable decentralized scheduler that takes into account time invariants and their resulting flaws into account to represent transportation tasks. Each task has to be able to be synchronized with n possible nodes for later transportation. Changes, e.g. time switches, task adding, or cancellation have to be automatically distributed in the network to specific nodes which are affected, in a way to keep communication overhead to a minimum.

Secondly, a realistic, robust cost representation has to be determined that accounts for dynamic operational duration times into account. This has to be split into a fixed and a dynamic cost representation. The fixed costs is an assumption based on linear machine time models, while the dynamic cost model considers blurring in time as well as blocking and a transportation route overflow. Therefore, the dynamic model‘s cost reflect change

over time such that the online routing is capable of rerouting the transportation based on flows, resulting in extending the transportation time on the current route to ensure robustness.

The third main module covers the route finding and execution. To overcome the size and dynamically changes of the system, an opportunistic online partial routing seems promising for finding a route while considering graph changes and time invariants. This is first done by determining feasible candidates, and subsequently by selecting a candidate. The determination and selection, compared to other methods, has to be scalable while operating in a decentralized structure. Therefore, an autarchic method has to be developed that successively determines feasible candidates and finally selects the most promising routes. Each route does not have to reach the destination, such that partial routes are determined during transportation to overcome time invariants causing graph changes.

During execution, a robust online method needs to ensure a deadlock-free routing, especially when inverse flows occur. The dynamic changes in arrival and transportation times of containers cause time-shifts during movement, meaning that the schedule cannot be adhered to. The route execution is a local route synchronization based on an online schedule synchronization of neighbors, sequentially updating sequential its affected neighbor based a unicast method.

Because to the decentralized organization results in a lack of global state knowledge, a robust routing table must be implemented to ensure a straightforward route finding without requiring high communication or computation effort by considering graph parts that cannot lead to a successful route. Therefore, initially, each reachable node initially receives a message containing the static costs to the destination and the destination id for every destination. During run-time, the real transportation time is tracked and postponed unicast to each affected node n_i with real caused cost when the sink has been reached. This value is later used to determine the dynamic costs for route finding.

5 Concept for routing

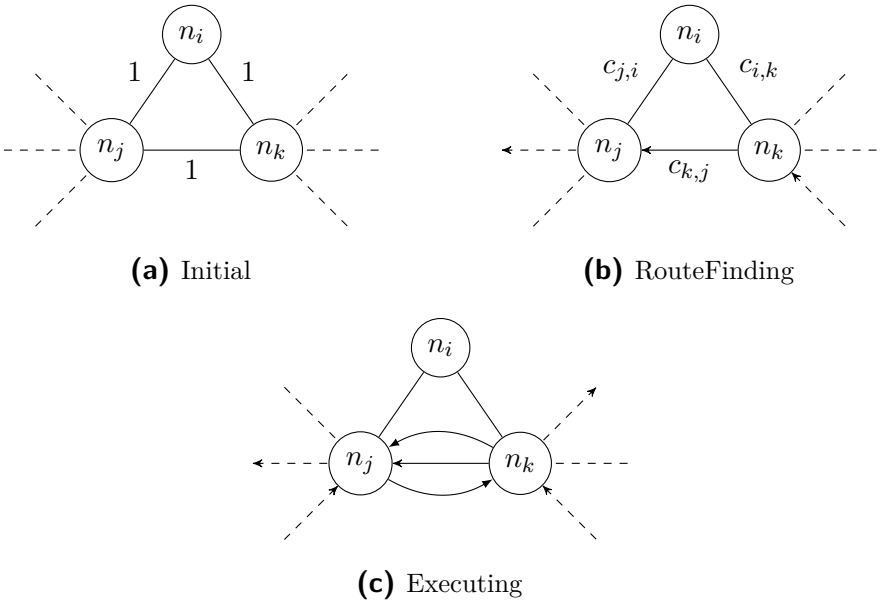
This chapter describes the routing protocol for transportation of an arbitrary number of containers in an automated and decentrally controlled complex material flow system independent in size. The requirements are finding a feasible path in linear time (scalability), ensuring a deadlock-free transportation time (robustness), and considering system changes over time (flexibility) of containers.

To ensure scalability of route finding in linear time and ensure a flexibility by quickly reacting to changes, the routing protocol operates opportunistically by determining partial paths online until the container has reached the destination. This is done by first determining a candidate set of all possible partial paths based on the current position of the container. Afterward, it uses a context- and congestion-aware metric to determine the candidate in the candidate set for routing. Finally, a time-slot is scheduled responsible for container handling at the intelligent device (PLC, see section 2.1)[194] at each TU. If during planning or executing a partial path system changes occurs that negatively affects this path negatively, e.g. blocking or possible deadlock, a rerouting is done to ensure system stability by planning and routing.

Deadlock-free transportation for all containers in the system is realized by considering the transportation task at each TU during candidate set determination. Each time-slot uses meta information for transportation cost calculation, which changes dynamically during the system operation. Furthermore, to ensure transportation during critical situations, a online deadlock detection and resolving method ensures a robust routing.

The overall transportation time is optimized by a load-balancing and knowledge-based cost function with forecasting, which reflects the dynamic behavior of containers during runtime. This is done by a stochastic model representing, determining possible transportation

[194]The type of PLC is irrelevant, e.g. IPC, SoftPLC, SlotPLC



(a) Initial **(b)** RouteFinding

(c) Executing

Figure 5.1: Exemplary operation visualization for executing container transportation.

blocking and deadlocks in the near future. Based on the computed cost, the shortest route in terms of time or distance is selected.

Fig.5.1 shows the three phases of determining a partial path and executing the transportation by the method to conduct a deadlock-free decentralized routing in a complex, decentralized network to provide scalability, flexibility, and robustness, as defined in section 2.3. The first phase is creating an operational basis that enables route finding and determination (see 5.1a). This is done by flooding the network from each possible sink and storing the hop count from each destination d to each intelligent device n_i. This generates a reachability matrix at each n_i as an initial route feasibility check to see whether d can be reached. Afterwards, the knowledge-based learning mechanism updates during run-time, based on existing and past flows and a future estimation of the costs between two devices n_i and n_j as $c_{i,j}$ and estimation to d as shown in fig. 5.1b. Finally, after a route has been chosen and starts being executed, each pair of n_i and n_j starts a handshake to secure deadlock-free transportation when an inverse flow from n_j to n_i exists. Prioritization for a container is done decentrally and executed as shown in fig. 5.1c.

This chapter is structured as follows. Section 5.1 describes the protocol structure which is used as an operational basis to reflect the system, to store the container, and

to execute the order. The used metric for load-balancing and knowledge-based cost computation is explained in section 5.2. Section 5.3 presents the opportunistic routing with forecasting to determine the candidate set of feasible routes based on the introduced metric. Subsequently, the chapter introduces the robust route execution is introduced which ensures a deadlock-free routing.

5.1 A flexible protocol structure for routing

A protocol structure is needed to ensure an error-free communication for route determination based on candidate set identification. Therefore, the protocol consists of a graph representation, a local task organization for container transportation-execution, and a well-formed structure for communication. The graph representation at each intelligent device consists of information to enable a proper routing. The local task representation is done by an encoded decentrally organized scheduler storing the necessary tasks for transportation on each TU. Finally, the well-formed structure for communication enables proper communication between intelligent devices for information collecting and spreading with a focus on routing.

5.1.1 Graph representation

To enable an operation to solve a specific issue with one algorithm, a fundamental basis is needed to represent the system. Particularly in the domain of a highly complex transportation system, a dynamic basis is necessary. It must represent changes over time which enable a reaction to improving the overall throughput in a dynamical manner during the system run-time.

Transportation system

The significant impact on the transportation time for one or more containers in a highly complex transportation system can be split into two elements: first, their physical, mechanical, and electrical wiring; and second, the organizational structure of TUs in a system. The physical structure is not considered in this work. It includes the motor organization, wiring, type selection, the electrical components, and their arrangement in

the wiring. The organizational structure is specified by the connection of TUs arranged to form a transportation system. As mentioned in section 2.1, to arrange the TUs, three main processes have to be considered. The first one is the bidirectional transportation of a container, which describes the ability to move the containers from one end to another and vice versa. The second process is the use of a fork for distributing containers over several conveyor lines. The third is the union, responsible for concatenating various conveyor lines into one. Positioning equipment (PE, a subset of TU) is able to provide the processes of fork and union respectively in time. This is given for example by a sorter distributing containers to different lines supplied by more than one line.

In a complex transportation system advances in time are crucial due to the dynamics in container movement. Each container position changes during runtime based on the planned route. This future position has to be considered in current route planning for incoming containers. Especially for bidirectional transportation, a TU is blocked in one direction for a period of time. To find a better[195] route, this fixed transportation direction in that specific period of time must be considered.

A time-varying graph[196] reflects the changes in the system over time by enabling structural dynamics in the model. This type of graph has its focuses on two operations: update and query. The update operation enables the insertion and deletion of vertices and edges during execution. In adaption to transportation systems, a bidirectional TU that moves a container in one direction over a period of time will change the edge orientation between two vertices. Querying focuses on finding a path for the container by determining a route during run-time considering the changes over time in edges and vertices. These changes impact the number of edges and vertices.

The following definition is used to determine a dynamic graph for container transportation in complex systems. Time marches in discrete time steps and is defined by $\tau = 0, \dots T$ with τ referring to the current time in the system. A dynamic network[197] $G[0,T]$[198]:= $\{G_\tau = (V_\tau, E_\tau)\}_{\tau=0}^{T}$ is a set of snapshots[199] $G_\tau \in G[0,T]$ with discrete time steps τ[200] and an underlying graph defined as $G_u = (V_u, E_u)$. The finite set V_τ contains all vertices

[195] Better in terms of shorter transportation time.
[196] Further called dynamic graph.
[197] A network is a weighted digraph.
[198] Adapted from [BYJB14], who define this type of graph for packet routing in complex networks without weights.
[199] One snapshot is one graph for a specific time.
[200] Subsequently, mathematical annotations related to time are written in Greek letters.

respectively to the time τ representing the PEs in the system capable of the fork and join processes and $E_\tau = (e_i, e_j, \ldots, e_n)$ as the set of edges at time τ with $e_i = (V_i, V_j)$ as an edge connecting two vertices V_i and V_j. Each bidirectional TU between two TUs which is able to fork and join containers is summarized[201] in one edge defined by $e_i = (V_u \times V_u)$. An edge e_i is a 2-tuple describing a flow by the underlying time independent, undirected graph. For $e_i = (V_i, V_j)$, the tail is defined as $\text{tail}(e_i) = V_i$ and the head as $\text{head}(e_i) = V_j$. Furthermore, in dynamic directed graphs, e_i is an ordered pair (2-tuple) $(i, j) := \{i, j \mid i \neq j\}$ representing a directed vertex from edge V_i to V_j. An inverse edge[202] (transportation flow) is defined as $e_\tau^{-1} = (i, j)^{-1}$ for an edge e_τ with the opposite direction from edge V_j to V_i. This leads to the definition of a deadlock in time-varying directed graphs. Each edge e_i is assigned a weight w_i. Weights are values defined as $w_i \in \mathbb{R}_{\geq 0}$ for edge i. A transportation system configuration[203] for one snapshot G_τ is deadlock free when eq. 5.1 is satisfied, notated by $free(G_\tau)$.

$$\text{free}(G_\tau) := \forall e \in E_\tau : e \in E_\tau \oplus e^{-1} \in E_\tau \tag{5.1}$$

Thus, no inverse flow exists for one edge $e_\tau(i, j)$ from one vertex $i \in V_i$ to vertex $j \in V_j$ for a specific time τ. If all snapshots in G fulfill eq. 5.1, the whole system is free of local deadlocks, illustrated in eq. 5.2.

$$\forall \tau \in T : \text{free}(G_\tau) \tag{5.2}$$

Analogous to transportation systems, a deadlock occurs when two containers are scheduled on one or more connected TU for container transportation in opposite directions during one specific period of time.

Figure 5.2 shows a transportation system with six TUs and six PEs (turn tables) are shown. Both TUs are summarized to one edge $E_\tau = (u, v)$ for time τ between the nodes $u, v \in V_\tau$ representing the PE. The equivalent snapshot for fig. 5.2 is $G_\tau = (V, E)$ with $V_\tau = \{v1, v2, v3, v4, v5, v6\}$ and $E_\tau = \{\{v_1, v_2\}, \{v_3, v_2\}, \{v_2, v_4\}, \{v_5, v_4\}, \{v_4, v_6\}\}$. As illustrated in fig. 5.3, the edge direction differs in time based on the transportation

[201] Bidirectional TUs degree of freedom is restricted to transporting a container forwards or backwards. Therefore, for the sake of simplicity a simple logic is sufficient as long as the input and output vertices control the flow of such TUs .

[202] As defined in [BL95]

[203] A configuration is a characteristic composition of different TUs to enable transportation tasks.

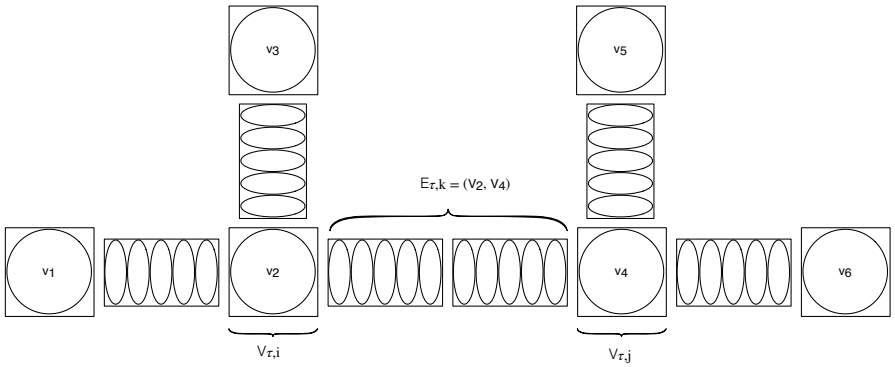


Figure 5.2: Example for a transportation system and its related nodes and edges in one graph.

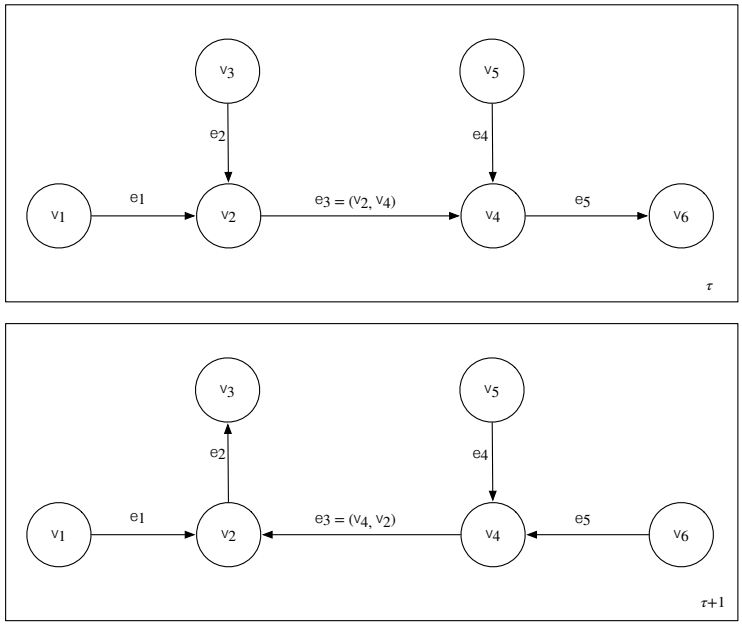


Figure 5.3: Two graphs G_τ and $G_{\tau+1}$ representing one transportation system in two discret time steps.

tasks for containers currently in the system. An update on the graph G_τ leads to a new graph $G_{\tau+1}$ with the edge set $E_{\tau+1} = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_4, v_2\}, \{v_5, v_4\}, \{v_6, v_4\}\}$. This enables enough flexibility to represent physically changes over time in the system. Therefore, a path formalization is needed to represent a path with different graphs successively advancing in time.

Utility based route representation

Besides the local control of TUs, enabling transportation physically for a container, a route has to be selected on the material flow level to control each TU in fully automated material handling systems. Due to their adaptive and decentralized structure, TUs require at least the information about the next target (TU) for one specific arriving container to handle properly.[204] Knowing the next TU for one container enables feasible transportation based on a prior route determination. Thus, decentralized route representation reduces the communication effort for transportation execution. Therefore, each TU stores the next hop for each passing container. This leads to hop-to-hop transportation without communicating with neighbors based on a previously determined route.

Each transportation task at one TU is a unique mapping between a route and one container. A route for one container consists of several trails[205] in a graph with edges advancing in time to represent a transportation task for one container. To enable routing in dynamic graphs, timed trails[206] are needed. Timed trails distinguish themselves from regular trails by mapping edges with a time respective to the dynamic graph describing the expected arrival time of the container at the TU. A timed trail r_c as in eq. 5.3 is a set of 3-tuple $s = E \times T \times T$ for a container c and r_c expressed as a set of 3-tuple $r_c := \{(e_i, \alpha, \beta), (e_j, \gamma, \delta), \ldots, (e_k, \psi, \omega)\}$.

$$r_c = \{(e, \alpha, \beta) \mid e \in E, \alpha \in T, \beta \in T, \alpha < \beta\} \tag{5.3}$$

A triple $s = (e_1, \alpha, \beta)$ is called a transportation slot in a trail r_c, and stores a transportation direction $e_1 \in E$ with the starting time *alpha* and an ending time β. Due to the consecutive execution of discrete time steps, every edge in a transportation slot in r_c has to advance in time. Each subsequently edges $e_i \in E_\tau$ and $e_j \in E_\tau$ in r_c have to be connected such that eq. 5.4 is fulfilled.

$$\{e_1, \ldots, e_n\} := \{e_i \neq e_j, \text{head}(e_i) = \text{tail}(e_{i+1}), \forall 1 \leq ij < n\} \tag{5.4}$$

[204] Properly in terms of deadlock-free transportation to the target.

[205] As defined in ([Wil96, p. 27]) determined during runtime of the system (as explained in 5.3). Each trail in one route defines the sequential transportation task such there is a fluent transition between two connected trails. A trail itself defines the executional task for one TU with an expected arrival time of the container.

Each MHS can be considered as a dynamic graph as explained previously. In dynamic graphs the set of vertices and/or edges can differ during runtime respectively to the time. Therefore, the goal is to find a trail. A trail or sometimes called simple walk, is a walk without duplicate vertices.

[206] Further called trail.

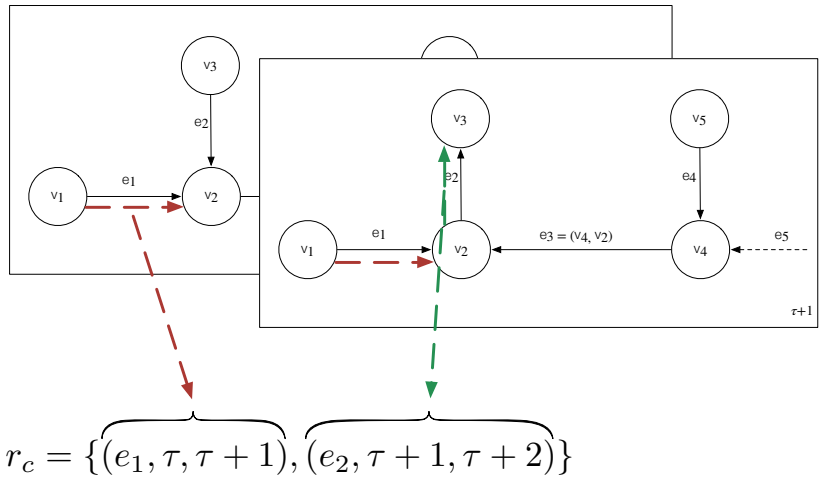


Figure 5.4: Time trail example.

Fig. 5.4 shows a trail intersecting two snapshots. In the time period $[\tau, \tau+1]$ edge e_2 changes its direction and enables deadlock-free transportation for container c. This direction change of edge e_2 is caused by the routing algorithm, explained later in section 5.3, querying to find the shortest path by using an update function on dynamic graphs to redirect edges by possibly inverting their direction from $e_i = (n_i, n_j)$ to $e_j = (n_j, n_i)$, such that at time τ, e_i is removed and e_j injected and usable. The stored information in the trail is used to identifying the transportation task for one container. Therefore, the trail r_c consists of two direct subsequently time steps, which is not necessary as long as the time restriction in eq. 5.3 is valid.

To increase the throughput of a transportation system, the simultaneous transportation of one or more container must be considered. Each container c is mapped to one trail r_c representing the transportation path. To secure deadlock-free transportation, each trails have to be checked for inverse flows. Therefore, a trail r_k towards r_l for container $k, l \in C$ is safe for transportation when no edge exists with an inverse flow during transportation.

$$\text{safe}(r_k, r_l) := \forall e_\alpha \in s_i \in r_k \cup r_l : e \oplus e^{-1} \in s_j \in r_l \tag{5.5}$$

Eq. 5.5 determines whether an inverse flow exists between two trails by comparing each edge in each slots to see whether they are safe for transportation. This equation is derived from eq. 5.1.

The waiting time between two slots $r_1 = (e_1, \alpha, \beta)$ and $r_2 = (e_2, \gamma, \delta)$ is computed by the function

$$\text{wait}((e_1, \alpha, \beta), (e_2, \gamma, \delta)) := \gamma - \beta \tag{5.6}$$

and transportation time for one slot is computed by the function

$$\text{trans}(e_i, \alpha, \beta) := \beta - \alpha \tag{5.7}$$

By combining both equations the overall transportation time is computed as follows

$$\text{duration}(r_c) := \sum_{i=0}^{n-1} (\text{trans}(s_i) + \text{wait}(s_i, s_{i+1})) \tag{5.8}$$

For current continuous transportation systems, eq. 5.5 is sufficient to enable a deadlock-free transportation. Each snapshot G_τ is valid until all transportation tasks are executed. This decreases the flexibility in creating new snapshots with higher transportation throughput. Consider transportation for container c with trail $r_c = \{(e_1, \alpha, \beta), (e_2, \ldots), (e_4, \ldots), (e_6, \ldots)\}$ in fig. 5.3 and a trail $r_i = \{(e_5^{-1}, \ldots), (e_4^{-1}, \ldots)\}$ for container i. If r_c is scheduled for transportation first the snapshot G_τ will be frozen[207] until the transportation has been executed. Even if container i with trail r_i transportation time is shorter, thus $\text{duration}(r_c) < \text{duration}(r_i)$, and no deadlock is given, r_i has to wait until c transportation has been executed.

To enable transportation with time-windows without considering a deadlock handling mechanism, and without resolving or preventing deadlocks during run-time, a trail has to be checked against all trails in the same time period. If an inverse flow exists in an overlapping time window, a deadlock situation is likely to occur during run-time of the system. Let $r_c = \{(e_1, \alpha, \beta), \ldots (e_n, \psi, \omega,)\}$ be a new trail for a container c to be considered for transportation. Then a time collision between r_c and r_i is defined as:

$$\begin{aligned} \text{timeColl}(r_c, r_i) := \forall((e_i, \alpha, \beta) \in r_c) \exists((e_j, \gamma, \delta) \in r_i) \\ : (\alpha \leq \gamma \leq \beta) \vee (\alpha \leq \delta \leq \beta) \end{aligned} \tag{5.9}$$

[207] A frozen snapshot is a graph which cannot be changed and no advances in time are possible.

$\text{safe}(r_c, r_i)$	$\neg\,\text{timeCol}(r_c, e_i)$	$\text{timedSafe}(r_c)$
0	0	0
0	1	1
1	0	1
1	1	1

Table 5.1: Trail constellation for deadlock occurrence.

Afterwards, each edge $e \in r_c$ has to be checked against inverse flows respective to their time such that:

$$\begin{aligned} \text{timedSafe}(r_c) := \forall((e_i, \alpha, \beta) \in r_c)\forall(r_j \in R)\nexists((e_i^{-1}, \gamma, \delta) \in r_j) \\ : (c \neq i) \wedge (safe(r_c, r_i) \vee \neg\,\text{timeCol}(r_c, e_i)) \end{aligned} \tag{5.10}$$

In summary, when adding a trail r_c to the system, all other trails $r_i : (c \neq i)$ must be checked for a possible deadlock.

The system is deadlock-free when trail r_c is safe to all other trails already in the system. As displayed in table 5.1, a deadlock occurs only when a trail r_i is not safe, such that the inverse flow occurs in the same time period. A container can only be added to the transportation system when its representing trail fulfills the requirement of being time-safe, such that no deadlock will occur. Once a trail in one route has been found each TU stores the needed part to enable hop-to-hop transportation for one container. Each transportation slot $s_i \in r_c$ is transformed into a task for the local scheduler, as explained in the next section.

5.1.2 Local job representation with decentralized scheduler

A decentralized scheduler is the fundamental data structure on which all further algorithms will operate on. It organizes the requests and information to control containers and handles the transportation process of one iID [208] as a time-restricted resource.

[208] Each TU holds one intelligent device, e.g. PLC, see section 2.1.

Local Information for Behavior Representation

Local information is required for fast decision-making without the need for state synchronization between IDs. The structure consists of one scheduler organizing jobs to execute transportation tasks by using operations in a specific order. A transportation task on one ID is the movement of one container over the TU to another TU by executing specific operations. Specific operations are TU related, e.g. bidirectional conveyor: starting the belt with parameters to determine the belt speed. Operations are TU specific and not considered further; instead, they are assumed as given.

Each ID at TU $\mathfrak{m}_\mathfrak{i}$[209] runs a scheduler[210] $\mathfrak{S}_\mathfrak{m}$ with jobs on TU $\mathfrak{m}$. Each TU holds one ID[211] in the system. A TU or PE $\mathfrak{m}_i$ is folded to one edge $E_i \in G_\tau$ such that a connection between two PEs $\mathfrak{m}_j$ and $\mathfrak{m}_k$ exists on the field level. Furthermore, a node N_i is mapped to one PE $\mathfrak{m}_j$ by an $\sim$. This mapping is then written as $N_i \sim \mathfrak{m}_j$.

A job $\mathfrak{j}_{c,\mathfrak{m}}$ for a container c executed on TU $\mathfrak{m}_i$ is executed on one TU in the order of the container's release time with the following related attributes:

- The release date $\rho_{c,\mathfrak{m}}$ defines the time when the job is expected[212] to arrive and be ready for processing by TU $\mathfrak{m}$ for container c.
- The due date $\delta_{c,\mathfrak{m}}$ is the completion date for transporting a container c on TU $\mathfrak{m}$. Processing the job after its due date is possible but can result in system inconsistency. This system inconsistency is given when the next TU transports a container in the opposite direction towards the current operating TU.
- The expected processing time $\psi_{c,\mathfrak{m}}$ describes the estimated time the TU $\mathfrak{m}$ needs for transporting the container c.
- A flow $f_{c,\mathfrak{m}}$ is a 2-tuple with an inbound e_i and outbound e_j with $e_i \in E_\tau$ and $e_j \in E_\tau$ for a container c on a TU $\mathfrak{m}$.

[209] The fraktur alphabet is used for variables and functions related to tasks and operationson on one ID.

[210] A scheduler $\mathfrak{S}$ for a TU is a chain (totally ordered set, see (see [Sta11, pp. 280 sq.])

[211] E.g. PLC executes the software with the implemented operations, whereas TU executes the physically task e.g. for a conveyor belt: The energy is transformed from electric to kinetic energy by the motor. Afterwards the belt transmits the energy from the belt to the container (energy loss, e.g. friction, is not considered), which causes the container movement.

[212] (see [Pin12, pp. 14 sq.]) defines ρ as the date on which the job can be computed. Due to the dynamic behavior of transportation, this time cannot be strictly adhered to, e.g. because of congestions during transportation.

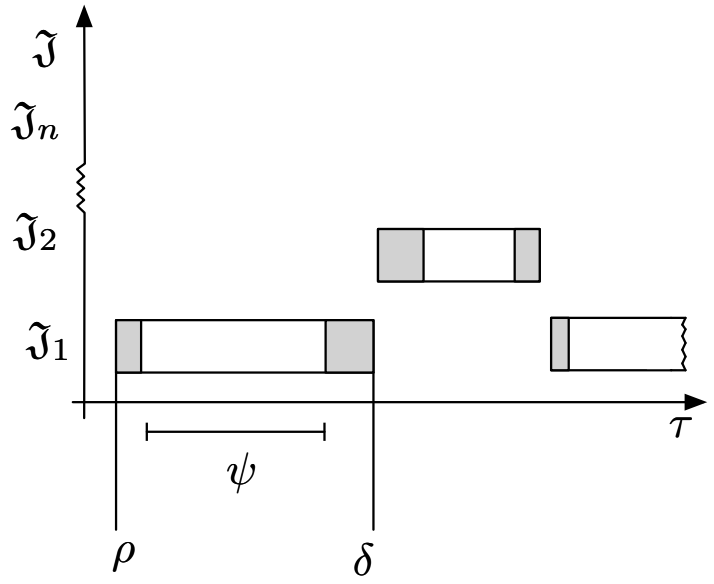


Figure 5.5: Example of tasks with operations for container transportation.

- e_i is the inbound handing over a container to the current TU. If e_i is a summarized edge of several TUs, e_i is unfolded and split across all bidirectional TUs such that $(i, j) = ((\mathfrak{M}_i, \mathfrak{M}_k), (\mathfrak{M}_k, \mathfrak{M}_l), \ldots, (\mathfrak{M}_m, \mathfrak{M}_j))$ with $i \in N$ and $j \in N$.
- The outbound e_j refers to the next TU the container will be handing over. Like e_i if e_j is a summarized edge, it is unfolded to point to the neighbor to simplify the operation.[213]

Fig. 5.5 shows three time-discrete jobs for executing three transportation tasks. Each job $\mathfrak{j}_{c,\mathfrak{m}}$ has an idle time:

$$\text{idleTime}(\mathfrak{j}_{c,\mathfrak{m}}) = \delta_{c,\mathfrak{m}} - \rho_{c,\mathfrak{m}} - \psi_{c,\mathfrak{m}} \tag{5.11}$$

in which the execution of the job $\mathfrak{j}_{c,\mathfrak{m}}$ can be delayed without violating the due date. Furthermore, a job's release date $\rho_{c,\mathfrak{m}}$ can be reduced safely to a minimum as the completion date $\delta_{i,\mathfrak{m}}$ of the predecessor, and the completion date $\delta_{c,\mathfrak{m}}$ can be extended up to the release date $\rho_{j,\mathfrak{m}}$ of the successor.

A time violation of one job occurs when the due date is missed. This can cause a time shift for all successive tasks until the violated time can be neutralized. The completion time

[213] Thereby, each ID only needs to point to the transportation direction.

for one job $\mathrm{j}_{c,\mathfrak{m}}$ is $\kappa_{c,\mathfrak{m}}$. The lateness[214] is noted by the function: $\mathrm{late}(\mathrm{j}_{c,\mathfrak{m}}) = k_{c,\mathfrak{m}} - \delta_{c,\mathfrak{m}}$ for job $\mathrm{j}_{c,\mathfrak{m}}$. If $\mathrm{late}(\mathrm{j}_{c,\mathfrak{m}}) > 0$, a violation has occurred. A time shift of a future task appears if $\kappa_{c,\mathfrak{m}} > \rho_{i,\mathfrak{m}}$ because the current task is not finished before the start-time of the next task. The buffer time between two succeeding jobs $\mathrm{j}_{c,\mathfrak{m}}$ and $\mathrm{j}_{i,\mathfrak{m}}$ is defined by: $\mathrm{buffer}(\mathrm{j}_{c,\mathfrak{m}}, \mathrm{j}_{i,\mathfrak{m}}) = \rho_{\mathrm{j}_{i,\mathfrak{m}}} - \delta_{\mathrm{j}_{c,\mathfrak{m}}}$.

Direct neighbor information for coordination

Due to the dynamics in complex systems, the meeting of one due date for a container‘s transportation on one TU can not be guaranteed. Therefore, coordination between direct neighbours is needed to ensure a proper transportation without causing deadlocks and invoking or reducing congestions between containers. Besides the trivial case of searching and executing the related job for the container, congestion awareness has to be considered. Moreover, local deadlock avoidances between two or more TUs is crucial to ensure an operating transportation system. To overcome these problems, the TU has to provide the operations “Wait” and “Move” and the possibility to handshake with its direct neighbors.

The Wait operation on one container invokes the transportation over the TU and lets the container wait in front of it. This operation on one TU gives the freedom to choose another container to transport first to solve a possible deadlock. Transporting one container over a TU is done by the Move operation. This essential operation is responsible for transporting the container based on the previously defined route.

To enable the Move operation, the information concerning in which direction to move must be stored locally in the scheduler. Therefore, a transformation and split operation are needed to distribute the defined trail over the TUs . Each element in the trail holds one job for one container on one TU, which reduces the communication overhead for routing to a local search on the scheduler for the specific job related to the container. When a container arrives at one TU, a search is triggered to determine the related job for that specific container. The search for the correct job for the container is a linear search in the set of all jobs and is done in $\mathcal{O}(n)$ with n as the number of jobs stored in the scheduler.

[214] See [Pin12], when positive the job is late and negative it finished earlier.

Consider a trail r_c for a container c as $r_c = \{(e_1, \alpha, \beta), (e_2, \psi, \omega)\}$. The underlying graph $G_u = V_u, E_u$ consists of three vertices and two edges. Assume no vertex $V_i \in N_u$ is considered to be error-prone, so no breakdown for one TU exists. Thus $N[0, T] := \{N_u\}_0^T$ is given. The transportation system consists of at least three TUs defined as: $\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3$ Each transportation slot in r_c has to be split and transformed into one job on each decentralized scheduler executed on one TU. Therefore, every slot $s_i \in r_c$ is transformed into two jobs: one for each TU represented by the node.

Algorithm 1 Split trail and convert to a job for scheduler at $\mathfrak{m}_i$

Require: $r_c \neq \emptyset \wedge \exists! e_l \in s_j : (\text{tail}(e_l) = \mathfrak{m}_i \vee \text{head}(e_l) = \mathfrak{m}_i)$

```
1:  function SPLIT(r_c, m_i)
2:      j_{c,m_i} ← (ρ_{c,m_i}, δ_{c,m_i}, ψ_{c,m_i} = 0, e_in, e_out)        ▷ Empty job with cost 0
3:      for all ⟨e_i, α, β⟩ ∈ r_c do    ▷ Linear search for transportation slot affecting m_i
4:          if head(e_i) = m_i then
5:              e_in ← tail(e_i)
6:              ρ_{c,m_i} ← β
7:          else if tail(e_i ∈ s) = m_i then
8:              e_out ← head(e_i)
9:              δ_{c,m_i} ← α
10:             ψ_{c,m_i} ← cost(e_in, e_out)          ▷ For cost function see sec. 5.2
11:             return j_{c,m_i}
12:         end if
13:     end for
14:     return j_{c,m_i}                ▷ Source or target have a cost(e_in, e_out) = 0
15: end function
```

By successively splitting every transportation slot $s_i \in r_c$ with the function split(r_c, m_i) for each TU in the trail (see algorithm 1), every TU in the trail is connected to execute the transportation of the container c. Every TU in the trail can be reached by the communication protocol explained in the next section.

Because of the dynamics in transportation (see section 2.1), meeting the due date of job j_{c,m_i} for moving one container over one TU cannot be guaranteed. To overcome a possible deadlock situation, a handshake is made between TUs related to vertex v, u. If the edge $e_1 = (v, u)$ for the execution time τ exists in $e \in E_\tau$ it is blocked and can not be changed until the transportation is finished. Otherwise, if $e^{-1} \in E_\tau$ exists up to a specific time $\tau + 1$, the current job and all future tasks are rescheduled by the function late$(\mathfrak{j}_{c,\mathfrak{m}})$. The prioritization of the container, which is being transported first, is done in a first-come, first-severed order.

5.1.3 Communication structure for synchronization

In decentrally organized transportation systems communication[215] plays a crucial role. It allows state notification for synchronization for a specific task and finding a feasible route for transportation in a network. The main purpose of state notification is the coordination of transportation tasks, e.g. handshake to avoid deadlocks. A feasible route can be found by propagating one route-finding task message. Each TU contributes its local part for path finding. This allows a more dynamic route finding with heterogeneous types of TU, because each TU provides its possible operations for transportation, instead of having one central unit that has to be extended with every new operation entering the transportation system. Furthermore, hard real-time communication is needed in industrial environments. Each data package has to arrive during each hop on one TU in a specific amount of time. If the time cannot be met, the TU can be considered as off-line.[216] On the other hand, a high waiting time for a message (e.g. with route information) causes the container to wait at the source,[217] which can cause congestion for future containers entering the system.

To enable hard real-time communication a package structure is needed that allows periodical communication. Therefore, the data packages are expandable or reducible during each hop. Furthermore, the waiting time has to be specified until a response can be considered. Especially in complex systems with many communication entities, it is difficult to decide when no response exists to an inquiry for e.g. a trail. A decision has to be made whenever a trail can be expected, when there is communication failure, or when many entities cause a delay in communication. Fixed waiting times, like in related work, tend to cause unnecessary waiting time when the fixed time is set too high, or do not considering different trails when setting too low because some trail‘s communication takes longer. By using a dynamic waiting time, there is a trade-off between communication speed and reliability.

In the following, the protocol structure is first introduced to enable communication between each TU. Afterwards, the section explains the different communication operations

[215] See section 2.1. For better readability, data packages are being communicated or forwarded and only containers are being transported. Forwarding describes one or more hops to the destination, and communicated the whole process, from the source to the destination.

[216] No communication is possible with an off-line TU.

[217] E.g. at a handover position to the system, which results in an unloading stop of the truck.

for sending messages in the system between other TUs. Finally, it proposes a function for calculating the dynamic waiting time for a response.

Transportation unit and package structure for communication

To enable communication between different TUs, syntax and semantic is needed as well as the physical ability to send messages. The physical ability is already given by several techniques in the network domain and can be divided into hard-wired by cable, and wireless communication with industrial wifi. Characteristics for communication in industrial environments are reliability and communication speed. Each TU has to provide a communication interface to send and receive messages. The interface transforms the message into chunks and sends it over a communication device[218] to the next TU until the target is reached (hop-to-hop communication). Hardware[219] for sending and receiving is assumed as given. Further, the syntax used to structure the message is the open standard OPC UA.[220] To enable communication between different TUs, a communication protocol that defines the semantic is needed. This protocol defines the structure of information and its meaning given by defined types and their variables. The data package structure is dynamic in size, which enables specific messages to be added or removed during hops without losing information from other messages.

The transmitting of messages operates at the undirected graph G_u (as shown in section 5.1.1), which is the completely unfolded graph of $G[0,T]$. G_u is defined as $G_u = (V_u, E_u)$ with V_u as the set of all TUs and $E_u = V_u \times V_u$ as a pair representing a connection between two TUs and is in relation with the dynamic graph for routing by $\forall G_\tau \in G[0,T] : G_\tau \subseteq G_u$. Due to the much longer transportation time of a container in relation to the communication time for one hop, the latter can be neglected.

The synchronization between two TUs $\mathfrak{m}_\mathfrak{i}$ and $\mathfrak{m}_\mathfrak{j}$ is done by messages. A message from $\mathfrak{m}_\mathfrak{i}$ to $\mathfrak{m}_\mathfrak{j}$ holds a set of information I and a trail t_k describing the path to $\mathfrak{m}_\mathfrak{j}$. Transmission of messages is done asynchronously, which in turn leads to asynchronous execution of messages. To reach the destination $\mathfrak{m}_\mathfrak{j}$ from the source $\mathfrak{m}_\mathfrak{i}$, every TU $\mathfrak{m}_\mathfrak{i}, \ldots, \mathfrak{m}_\mathfrak{j}$ in the trail t_k receives and forwards the message. One forwarding to the next direct neighbor is called a hop. By forwarding the message successively to the next direct neighbor in t_k a

[218] Hardware capable of sending and receiving (See [G013, pp. 869 sq.]) messages.
[219] Regardless of whether it is hard-wired or wireless for communicating.
[220] OPC UA described in [HSK08] is an industrial standard for a message structures for communication in industrial environments, and can be executed on IDs.

hop-to-hop transmission is done. The message arrives as long as $\mathfrak{m}_\mathfrak{i}$ is connected by the elements in t_k to $\mathfrak{m}_\mathfrak{j}$. A recalculation of t_k is not considered, therefore if $\mathfrak{m}_\mathfrak{k} \in \mathfrak{s} \in \mathfrak{r}_\mathfrak{k}$ goes off-line, the message can not arrive at $\mathfrak{m}_\mathfrak{j}$ and being executed.

If no trail t_k exists, a multi-cast is done by flooding the network. Each direct neighbor receives a duplicate of the message expect the source to ensure a loop-free trail. Receiving the message, the TU adds itself to the trail and broadcasts a new duplicate to each direct neighbor except the source. When the target is reached, an acknowledge message is sent back by iterating the found trail backwards.

Dynamic waiting and response time

The waiting time for an answer is difficult to estimate in complex transportation systems by using hop-to-hop transmission. It relies on the computation time of the TU and the distance to other TUs . Hereby, distance is the number of hops the message takes to reach the target and is defined as distance$(\mathfrak{m}_\mathfrak{i}, \mathfrak{m}_\mathfrak{j}) := |t_k|$ with t_k as the trail between $\mathfrak{m}_\mathfrak{i}$ and $\mathfrak{m}_\mathfrak{j}$. Industrial networks compared to commercial networks[221] focus on the quality of the service and on hard real time capabilities, and are assumed to transmit data between 250μ and $10ms$. Therefore, the transmission time between two TUs $\mathfrak{m}_\mathfrak{i}$ and $\mathfrak{m}_\mathfrak{j}$ is assumed as constant and defined in the variable *TransTime*. Further, the computation time for handling a message is assumed to be constant too and defined in *CompTime*. The waiting time for one expected answer can be computed as follows:

$$\text{duration}(\mathfrak{m}_\mathfrak{i}, \mathfrak{m}_\mathfrak{j}) = (\text{distance}(\mathfrak{m}_\mathfrak{i}, \mathfrak{m}_\mathfrak{j}) * (\mathit{TransTime} + \mathit{CompTime})) * 2 \tag{5.12}$$

To reduce the number of considerable paths, a hop limit *HopLimit* and *HopCounter* is used. It limits the maximum size of t_k because $\mathit{HopCounter} = \mathit{HopLimit}$ in the beginning, the decreases with every hop until it reaches zero. In this situation $\mathit{HopLimit} \leq 0$, the message is discarded and not further duplicated. A maximum waiting time *MaxWaiting* is used to react if the destination is off-line. If *MaxWaiting* is too small, e.g. $\mathit{MaxWaiting} \leq$ duration$(\mathfrak{m}_\mathfrak{i}, \mathfrak{m}_\mathfrak{j})$, the path between $\mathfrak{m}_\mathfrak{i}$ and $\mathfrak{m}_\mathfrak{j}$ is never found. In contrast, if *MaxWaiting* is too high[222], the TU used as a source is blocked for *MaxWaiting* before the system can react to an off-line TU. Due to the flooding, several answers for one request are possible, e.g.

[221] (See [G013, p. 281])

[222] High is project specific and depend on the used transmission technique and structure of the transportation system.

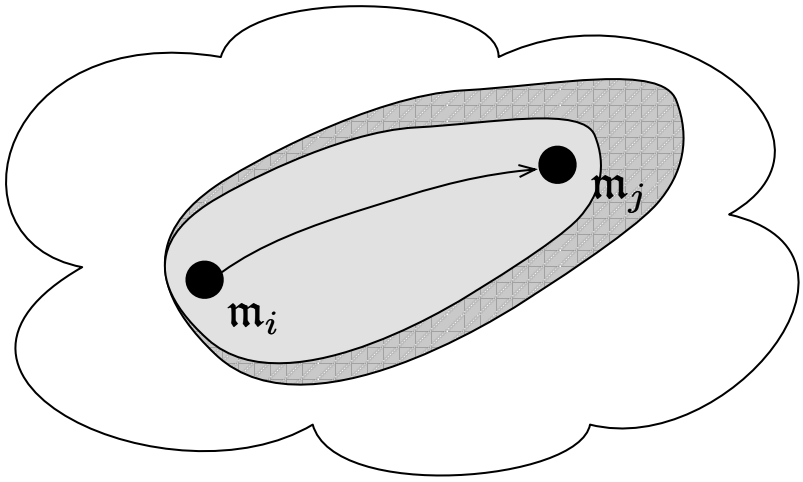


Figure 5.6: Repercussions on solution area (striped area) and evaluable solutions (gray area) controlled by *MaxWaiting* and *HopLimit*.

.

when requesting for a route, due to the broadcast, several possible routes are determined. To decrease the waiting time and reducing the number of answers, *WaitingTime* is used. After the arrival of the first answer, the TU waits *SolutionWaitingTime* time units for further answers. The set of answers is then evaluated, and one candidate is chosen for execution.

Fig. 5.6 shows the repercussions on the number and characteristics of the solutions considered for routing are shown. The cloud represents the transportation system. First, the reduction is done by limiting the path length with *HopLimit*, represented by the stripped area. Because a path search message is dropped when the *HopCounter* reaches zero, paths with greater lengths can not be found. The second reduction is given by the *MaxWaitingTime* such that no waiting time for off-line elements exists. It is assumed that shorter paths to the destination respond more quickly than longer paths, because of the assumption of static communication time between the TUs . Shorter paths consider fewer TUs in for communication.

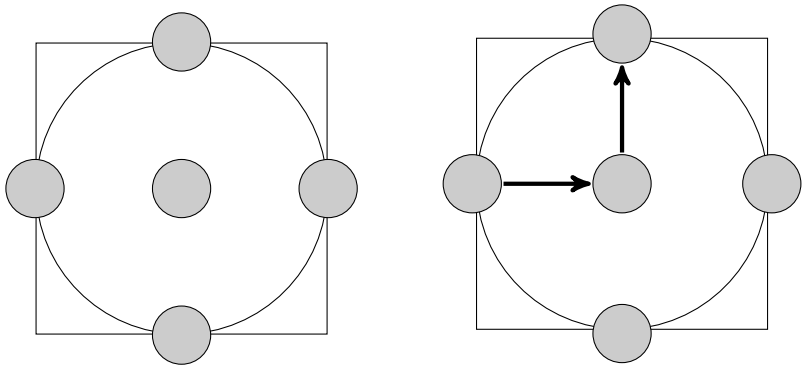
5.2 Transportation Unit evaluation under global state uncertainty

TU evaluation and selection to determine a path have a high impact on the overall transportation times and efficiency of the system. On a high level, few selections are needed to create a respective path for each container. If intersections exist, paths may interfere and result in blocking, which can lead to a system with parts not capable of moving because TUs are in a deadlock - state. Overloaded areas in the system result in blocking and longer transportation time than estimated during route finding. At the low level, the transportation time at each TU for one specific container has to be estimated. This time is defined by the physically model approximating the TU‘s transportation capabilities. Therefore, the metric is capable of load balancing to overcome blocking, and of estimating future inverse flows by synchronizing schedulers with current information at a vertex when a path finding message is triggered.

This is done by determining and evaluating each element. The determination is based on the communication abilities of TUs. Every reachable TU based on the dynamic waiting and response time is considered and evaluated. The evaluation considers the local scheduler information only and synchronizes the TUs after a route has been selected. Based on the evaluation a path can be selected, and the routing for one container triggered. The evaluation of one TU is done in two steps. The first is a static cost model approximating the mechanical capabilities based on a physical model for each TU type in the transportation system. The second is a dynamic model deriving short future transportation time by considering historical data and future jobs stored in the scheduler.

5.2.1 Static cost model

The static cost model allows a prompt computation and evaluation of a TU for path finding when no historical data exists. Particularly, in the starting phase of the transportation system, assumptions must be made about the transportation time of each TU. These assumptions are split into a fixed processing time for each possible operation and a learning approach based on historical data to estimate future waiting times.



(a) Initial TU State

(b) Local Path with c entering left and leaving top exit

Figure 5.7: Exemplary visualization of initial state (left) and one local path for moving c from left to top side.

As a basis, each TU is a local state machine.[223] A TU can only be in one state at one point in time and for a given duration. After this duration, a state transition can occur, or the same state can be executed again. This state representation is based on the state modeling and programming of TU motor control, which executes states by receiving sensor data and sending motor control data in the prescribed order given by the PLC program (see section 2.1). A set of ordered states with a predefined order of execution is called an operation.

Processing-time

Each transportation task has to be executed physically. This is done by transforming electrical into kinetic energy using a motor. The motor control is done by the engineering programming the PLC, which sends analog or digital (depending on the motor interface) parameters. When the parameters are sent to the motor, it starts transforming data into a specific amount of voltage, which produces the kinetic energy. The processing time for one TU $\mathfrak{m}_i$ is defined by p_{ij} executing the operation $\mathfrak{o}_j$.

To compute p_i linked points are used. A set of linked points is one local transportation path $\mathfrak{l}$ for TU $\mathfrak{m}_i$ and represents one transportation task from one end to the other end on one TU. Each operation is related to one transportation job as displayed in fig. 5.7.

[223] Local in adaptation to local control and, from a smaller point of view, in the entire system.

Therefore, the static processing time on TU $\mathfrak{m}_i$ with operation $\mathfrak{o}_j$ is computed by:

$$\frac{d_j}{v_i} + \frac{v_i}{a_i} \tag{5.13}$$

with d_j as the transportation distance for one operation $\mathfrak{o}_j$ by:

$$d_j = \mid \sum_{m=0}^{n} l_m \mid \tag{5.14}$$

and v_i as the velocity and a_i as the acceleration representing the motors capabilities on TU $\mathfrak{m}_i$ to move a container c.

Due to the dynamic behavior in transportation caused by blocking, the transportation time can not be simplified by d_j only. Therefore, an empirical method is used to store and evaluate with a maximum of $1,000$[224] samples on one TU for each operation. Every operation $\mathfrak{o}_i$ has to be evaluated separately because large time differences can occur (peaks), e.g. in fig. 5.7 shows that straightforward transportation is much faster than around-the-corner transportation. A more feasible way to compute the processing time p_{ij} is:

$$p_{ij} = \begin{cases} \frac{d_j}{v_i} + \frac{v_i}{a_i} & \text{if} |M_i| = 0 \\ M_{ijk}, k = \frac{n}{2}, n = |M_i| & \text{if} |M_i| \bmod 2 = 1 \\ M_{ijk}, k = \left\lfloor \frac{n}{2} \right\rfloor, n = |M_i| & \text{if} |M_i| \bmod 2 = 0 \end{cases} \tag{5.15}$$

with M_i as the set of all samples for operation $\mathfrak{o}_i$. A sample is created by recording the transportation time of one container over the TU with one operation. The waiting-time is excluded to distinguish between transportation and waiting time. Waiting time is caused by other transportation tasks in the system and can be avoided by detours. On the other hand, transportation on TUs is mandatory, and a selection of TUs must be made for transportation based on their execution time to find a proper path. By distinguishing between processing time and waiting time, the path finding method is able to identify slow TUs due to overloaded paths, and can therefore able to react with a detour to reduce the load at specific paths.

[224] After 1,000 data points the 95 % confidence level does not change significantly, see [Isr09].

Waiting-time

When blocking occurs, a state must be interrupted because it can not be executed further. Waiting time relates to ordered historical data W_i with the order $\leq$. Waiting time is measured during run-time because blocking is difficult to estimate without any transportation knowledge. Therefore, when $|W_i| = 0$, the median waiting time w at TU $\mathfrak{m}_i$ is $w = 0$. During system run-time, each waiting time duration is recorded and evaluated, such that w is the median of the set of recorded values. This leads to the following definition:

$$w_i = \begin{cases} 0 & \text{if} |W_i| = 0 \\ W_{ij}, j = \frac{n}{2}, n = |W_i| & \text{if} |W_i| \bmod 2 = 1 \\ W_{ij}, j = \left\lfloor \frac{n}{2} \right\rfloor, n = |W_i| & \text{if} |W_i| \bmod 2 = 0 \end{cases} \tag{5.16}$$

Because of the restrictive usage of storage[225] in PLC programs $|W_i|$ is limited to $1,000$. The median is used due to its low sensitivity to peaks produced by few containers with possible long waiting times.

Intermediate-state times

For a more precise evaluation of container handling on one TU, the processing times of intermediate states are considered. Such states ability the necessity to enable an error-prone execution of the successor state, e.g. lowering a lifter to receive a container. Intermediate states can be a precondition for executing one operation, and a postcondition for future states such that the TU has to be in a specific state. Pre- and postconditions are regular intermediate states except that they are executed before (precondition) or after (postcondition) one operation. In contrast to the states considered for computing the processing times, intermediate-states are not always considered if an TU is already in a proper state such that transportation can be executed immediately.

Intermediate-states are classified in terms of their urgency within execution as either mandatory or optional. Mandatory intermediate states always have to be executed during an operation to enable proper transportation. Such states are defined by a constant value

[225] See section 2.1. An array has to be fixed in size; dynamic allocation is not possible due to the determinism restriction defined for PLCs.

$s_i = \tau$ for operation i, with τ as the execution time for the intermediate-state. Optional intermediate-states are states which can be executed prior to transportation, between two transportation tasks, or the TU can only be in this state due to previous transportation. To compute their times, the previous job in the scheduler must be considered. Let $\text{pre}(\mathfrak{o}_i)$ be the precondition and $\text{post}(\mathfrak{o}_i)$ be the postcondition for one operation $\mathfrak{o}_i$ for one job i. Thus $\mathfrak{o}_{i-1}$ is the operation of the previous job and $\mathfrak{o}_{i+1}$ the operation of the successive job. Afterwards s_i can be computed as follows:

$$s_i = \begin{cases} 0 & \text{if } \text{post}(\mathfrak{o}_{i-1}) = \text{pre}(\mathfrak{o}_i) \\ \tau - \text{buffer}(i-1, i) & \text{else} \end{cases} \tag{5.17}$$

When the postcondition of the previous operation is equal to the precondition of the current operation, the TU is capable of transporting immediately. Otherwise, the TU has to execute an intermediate state to be able to properly transport the current container. The amount of time is therefore given by the available buffer time between the previous job and the current job representing the operation.

Static cost-model

The static cost model covers costs for operations possible without communication with the neighborhood and is therefore based on local information. To determine the static costs for transporting one container over one TU, the operation is evaluated to be capable of transporting the container over a possible path. The costs s_{ijk} for container i using the operation $\mathfrak{o}_j$ on TU $\mathfrak{m}_k$ is computed as follows:

$$s_{ijk} = p_{kj} + w_k + s_j \tag{5.18}$$

Therefore, s_{ijk} describes the time cost of transporting one container over the TU. This cost model does not cover the path-specific costs like the time to the destination and the likelihood of blocking during the transportation.

5.2.2 Dynamic cost determination

Besides the TU evaluation to determine the time needed to transport the container, the future path length and its costs also have to be estimated. Especially in dynamic systems

with partial information, a future path is difficult to determine. This occurs when transporting many containers in large systems with many intersections, which enables a larger degree of freedom in finding different paths from one source to a destination.

First, the following introduces an approach to estimated future paths by assuming costs for successive partial paths from the current TU to the destination. Afterwards, the first possible scheduling slot for the transportation on a TU is estimated to determine the transportation job. Finally, in synchronization with the slot search at TU, validation is done to ensure that transportation does not violate the system state.

Future path estimation

As described in section 3.4, the opportunistic routing algorithm at least needs the information concerning whether the current path is a dead-end or whether a exists a further path to the destination exists. The introduced approach uses a dynamically changing update function estimating the duration of one TU transportation task to every reachable destination. This is in contrast to current methods which using a fixed cost estimation, which does not reflect dynamic changes in transportation and tends to overload a specific path. This overloaded path is most likely the shortest path.[226] To determine whether a path is a dead-end or whether a further path to the destination exists, a routing table is created prior to the system start, as explained in section 5.3.1. During transportation, the time needed to transport a container c over the TU is stored. When a container c arrives at its destination, an update message is triggered, refreshing all times from each reachable TU to the current target by back-warding[227] the routing path. Each TU receiving the message computes the cumulative transportation time from the destination to its current position and stores the new time in the routing table to that specific destination. After computing the new transportation time, the local scheduler removes the job.

Consider a trail r_c for container c that has already been deployed at the destination. Each time c has been executed on TU $\mathfrak{m}_i$ and moved forward to the next TU $\mathfrak{m}_{i+1}$, a

[226] See [DSL14]
[227] Inversing the routing path, going from the destination to the source.

sample $s_{\mathfrak{m}}$ has been created. Then, the routing table for one $\mathfrak{m}$ is updated by the time $t_{i,j,k,\mathfrak{n}}$:

$$t_{i,j,k,\mathfrak{n}} = s_{ijk} + \sum_{\mathfrak{l}=\mathfrak{k}}^{\mathfrak{n}} s_{\mathfrak{l}} \tag{5.19}$$

with $\mathfrak{n}$ as the destination in r_c with $s_{\mathfrak{n}} = 0$ and a routing table $R_{\mathfrak{m}_i}$ at $\mathfrak{m}_i$ as a set of 2-tuple $(\mathfrak{n}, \tau)$ with $\mathfrak{n}$ as the destination at TU $\mathfrak{n}$ and the estimated time τ from $\mathfrak{m}_i$ to TU $\mathfrak{m}_n$ by a previous evaluated trail r_k, $k \in C$. If the entry $(\mathfrak{n}, \tau)$ already exists, it is overwritten with the new value. It is assumed that the new value represents the current system state more accurately than to the old one.

The overall estimated transportation cost done by a trail r_c is then:

$$estCost(r_c) := \{k = \text{tail}(\mathfrak{f}) \mid \mathfrak{f} \in r_c E \mathfrak{j}_c, E : r_c \to \mathfrak{j}_c\} : \sum_{i}^{n} t_{i,j,\mathfrak{k},\mathfrak{n}} \tag{5.20}$$

with the bitotal[228] relation E assigning one transportation slot in r_c to one transportation job $\mathfrak{j}_c$ on the scheduler. The real costs, the container transportation time, can only be calculated after transportation when the sampling has been done which is given by:

$$cost(r_c) := \{k = \text{tail}(\mathfrak{f}) \mid \mathfrak{f} \in r_c R s_k, R : r_c \to \mathfrak{s}_k\} : \sum_{\mathfrak{k}}^{\mathfrak{n}} s_{\mathfrak{k}} \tag{5.21}$$

Hereby, the bitotal relation R maps one TU $\mathfrak{m}_i$ in the trail given by the flow $\mathfrak{f}$ to one sampling.

Furthermore, it is assumed that between n parallel paths to one destination, an equilibrium[229] in transportation time will occur. Assume n containers exist with the same destination $\mathfrak{m}_n$ and have to be split at m parallel paths at the same time in the routing tables. Because of the sequential operation order was given by computational units like the industrial PLC, each determination of each trail is executed sequentially. In the beginning, the first container c is assigned the fastest trail r_c. Afterwards, the routing table is updated based on the determined samples $s_{\mathfrak{m}}$. The second container $c+1$ is then assigned a trail r_{c+1} with the lowest transportation time. If the updated sample results in lower values in the routing tables $\text{estimatedCost}(r_{c+1}) \leq \text{cost}(r_c)$ then the same trail $r_{c+1} = r_c$ is used. This trail is utilized until the values in the routing table are higher

[228] Every element in r_c has at least one or more partners in $\mathfrak{j}_c$ and vice versa.
[229] n parallel paths will split the container load equally if they have the same transportation time.

than the current values at the parallel paths given by estimatedCost($r_{c+1} > \mathrm{r_c}$), which is caused by the increasing waiting time the trail is overloaded. In this case, the next container $c + k$ is assigned a different trail r_{c+k} because it holds a lower time to the destination. Afterwards, the routing tables of the newly used parallel path with trail r_{c+k} are updated. Because the length and the processing time are equal, the only difference between the previous path and the new one is given by the higher waiting time, because the first one is overloaded. This path switching is done every time a parallel path is lower than the previous one until estimatedCost$(r_c + 1) \approx \mathrm{cost}(r_c)$. Then, the following load balancing holds: if $n <= m$ every container c uses a different parallel path; if $n > m$ the containers are being split equally between all parallel paths.

Linear time-slot determination

Based on the local costs $t_{i,j,\mathfrak{k},\mathfrak{n}}$ and flows defined in r_c utilized by the future path costs $\mathrm{cost}(r_c)$, the trail is transformed to jobs $\mathfrak{j}_{c,\mathfrak{m}_i}$ respectively on each TU $\mathfrak{m}_i \in s \in r_c$.[230] The transformation process interacts with other already transformed processes waiting for execution and deletion on the scheduler. This interaction results in time-shifts in the start- and end-times. Start-time is influenced by previous jobs processing a container, and end-time by the successive TU $\mathfrak{m}_{i+1}$ still processing a different container. To schedule the transportation of a container c the bitotal relation $E : r_c \to \mathfrak{j}_c$ transforms every transportation slot $s \in r_c$ into one job $\mathfrak{j}_{c,\mathfrak{m}_i}$ on one TU $\mathfrak{m}_i$ successively. This ensures deadlock-free scheduling by considering inverse flows during planning before fixing one job in each scheduler on every TU $\mathfrak{m}$ used in r_c by finding feasible transportation slots.

Let $G_u = \{V, E\}$ with $V = \{V_1, V_2, V_3, \ldots, V_n, T_t\}$ be a graph (see top fig. 5.8). And a trail r_i from V_1 to V_t and a trail r_j from V_n to V_1 for containers $i, j \in C$ already scheduled. Due to the inverse flow between r_i and r_j for e_1 two snapshots $G_\tau, G_{\tau+1}$ are needed to represent the flows. Then a trail $r_c = \{s_1, \ldots, s_t\}$ from V_1 to V_t is scheduled (transformed to jobs on every required TU $\mathfrak{m}_k \in s \in r_c$) by E using algorithm 2 with the parameters $(r_c, \mathfrak{m}_d, \alpha)$ with $\mathfrak{m}_d$ as the next TU for handover, α as the earliest starting time, and has the amount of jobs at $\mathfrak{m}_i$ as follows:

- The earliest scheduled time α is set to the current system time $\tau \in [0, T]$.

[230] s as a transportation-slot, defined in eq. 5.3

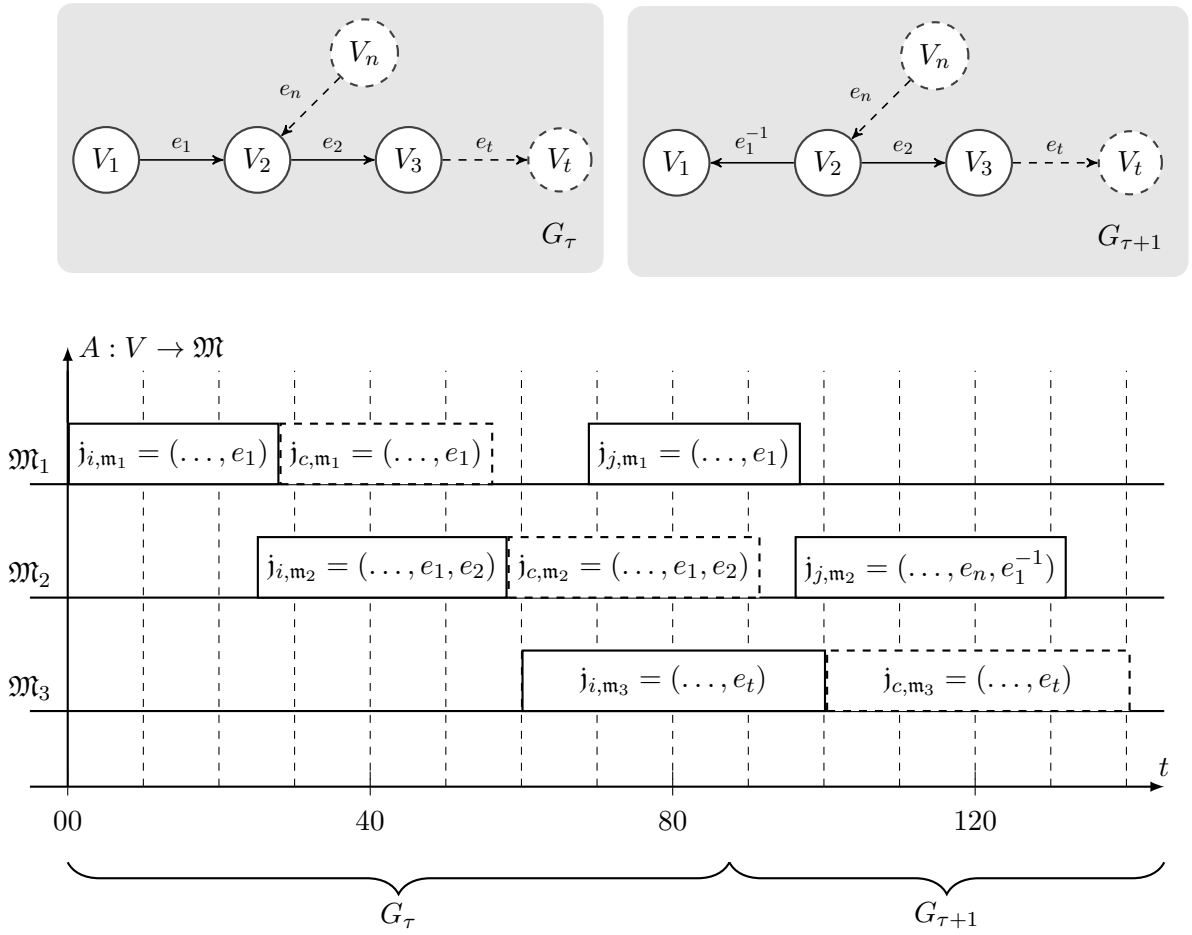


Figure 5.8: Changing snapshots during scheduling (top), scheduled jobs for one transportation task on three MHEs (bottom).

- When $n = 0$ (alg. 2 line 5) jobs exists on $\mathfrak{m}_k$ then the job starts at α as the first considerable time.
- If $n > 0$ (alg. 2 line 7-13), a free slot with at least buffer$(\mathfrak{j}_i, \mathfrak{j}_j) \leq \psi_k$ between two jobs $\mathfrak{j}_i$ and $\mathfrak{j}_j$ will be searched and scheduled in between.
- Otherwise, when no free slot with buffer$(\mathfrak{j}_i, \mathfrak{j}_j) \leq \psi_k$ exists (alg. 2 line 14), the job will be scheduled last $(\delta_n, \delta_n + \psi_k, \psi_k)$ with δ_n as the deadline of the last job.
- Last, $\alpha = \delta_k$ and algorithm 2 will operate on the next TU $\mathfrak{m}_{k+1} \in s \in r_c$ until the last job $\mathfrak{j}_n$ on $\mathfrak{m}_{k+n}$ have been scheduled.

Fig. 5.8 shows the effects of the scheduling for three TUs . The trail r_c has been split into three jobs, each job on one TU fulfills the transportation for container c. For TUs

Algorithm 2 First possible execution on TU $\mathfrak{m}_k$ for container c

```
1: function FindTimeSlot(r_c, m_d, α)
2:     ρ_k, δ_k, ψ_k                                  ▷ Not scheduled job
3:     ψ_k ← t_{c,j,m_k,m_d}
4:     if n = 0 then                                  ▷ n as the amount of jobs at m_k
5:         return (α, ρ_k + ψ_k, ψ_k)
6:     end if
7:     for all δ_l ≥ α do
8:         if ρ_{l+1} ≥ ρ_k + ψ_k then
9:             ρ_k ← δ_l
10:            δ_k ← ρ_k + ψ_k
11:            return (ρ_k, δ_k, ψ_k)
12:        end if
13:    end for
14:    return (δ_n, δ + ψ, ψ)
15: end function
```

$\mathfrak{m}_1$ and $\mathfrak{m}_2$, the buffer between two jobs is high enough so that a job can be scheduled in between. On TU $\mathfrak{m}_3$, the earliest execution time α is too high, causing a time shift of the third job $\mathfrak{j}_{c,\mathfrak{m}_3}$. This results in a waiting time of container c in front of $\mathfrak{m}_3$. The waiting time of c causes a time shift of job $\mathfrak{j}_{j,\mathfrak{m}_2}$ on TU $\mathfrak{m}_2$.

The scheduling of jobs influences the amount and characteristics of snapshot G_τ. Each job is linked to an operation by the defined transportation flow. This leads to updating the current snapshot G_τ, which represents the current state of the flows. E.g. by the necessity of an inverse flow e_1^{-1} for a specific amount of time, as shown in fig. 5.8 for job $\mathfrak{j}_{j,\mathfrak{m}_2}$, which leads to the snapshot $G_{\tau+1}$.

5.3 Decentrally controlled routing

A path for routing is needed to transport containers in decentralized highly complex transportation systems. Each path has to be found fast and without system status violation. Based on the protocol structure presented in section 5.1 for receiving and sending messages between TUs for state exchange a search for a path can be reserved and executed. The metric used in section 5.2 allows an evaluation of each TU, and later a selection to determine a path.

The structure of the used opportunistic routing first determines the reachability of the destination at each promising TU in the starting phase of the system. Based on the reachability data a candidate set is created storing all possible trails to the destination. Each candidate in the candidate set is evaluated and the most promising trail selected. Afterwards, the trail is split and scheduled on each TU involved by transporting one container based on the trail. During transportation, unforeseen dynamics can occur, e.g. container blocking. These occurrences result in time-shifts in the systems which have to be taken care of. Therefore, the execution of transportation uses a safety handshake method to ensure deadlock-free transportation when such occurrences occur.

This section first introduces the reachability table to determine dead-ends and estimate future path costs to the desired destination. Second, the section presents a candidate set creation method with feasible routes to the destination. Afterwards, the selection of the candidate is introduced, along with the decentralized route reservation method, which allows a parallel route reservation. Finally, the route execution is presented with local handling of deadlock occurrence and resolving based on unforeseen dynamics.

5.3.1 Opportunistic path exploration

In highly complex and dynamic transportation systems, the goal is to find a feasible and reliable trail r_c for a container c, such that c arrives at the desired destination with less blocking. The validity of the system then has to be guaranteed by avoiding deadlocks such that every subsystem is reachable. Furthermore, the routing method must be able to run in a fully decentralized way without full knowledge of the system and be able to be executed in parallel on distributed TUs .

Fig. 5.9 shows the processes operating on one TU to find partial trails to transport c to the destination. An opportunistic path exploration and routing method is used to find feasible partial trails r_c to the destination. First, a reachability analysis is conducted in the initialization phase, which computes the routing table R_i for each TU. Next, the routing phase starts by executing the computation cycle in steady state. This first executes the transportation operations stored in the scheduler and afterwards searches for r_c when a new container is about to arrive in the system or needs a further route.

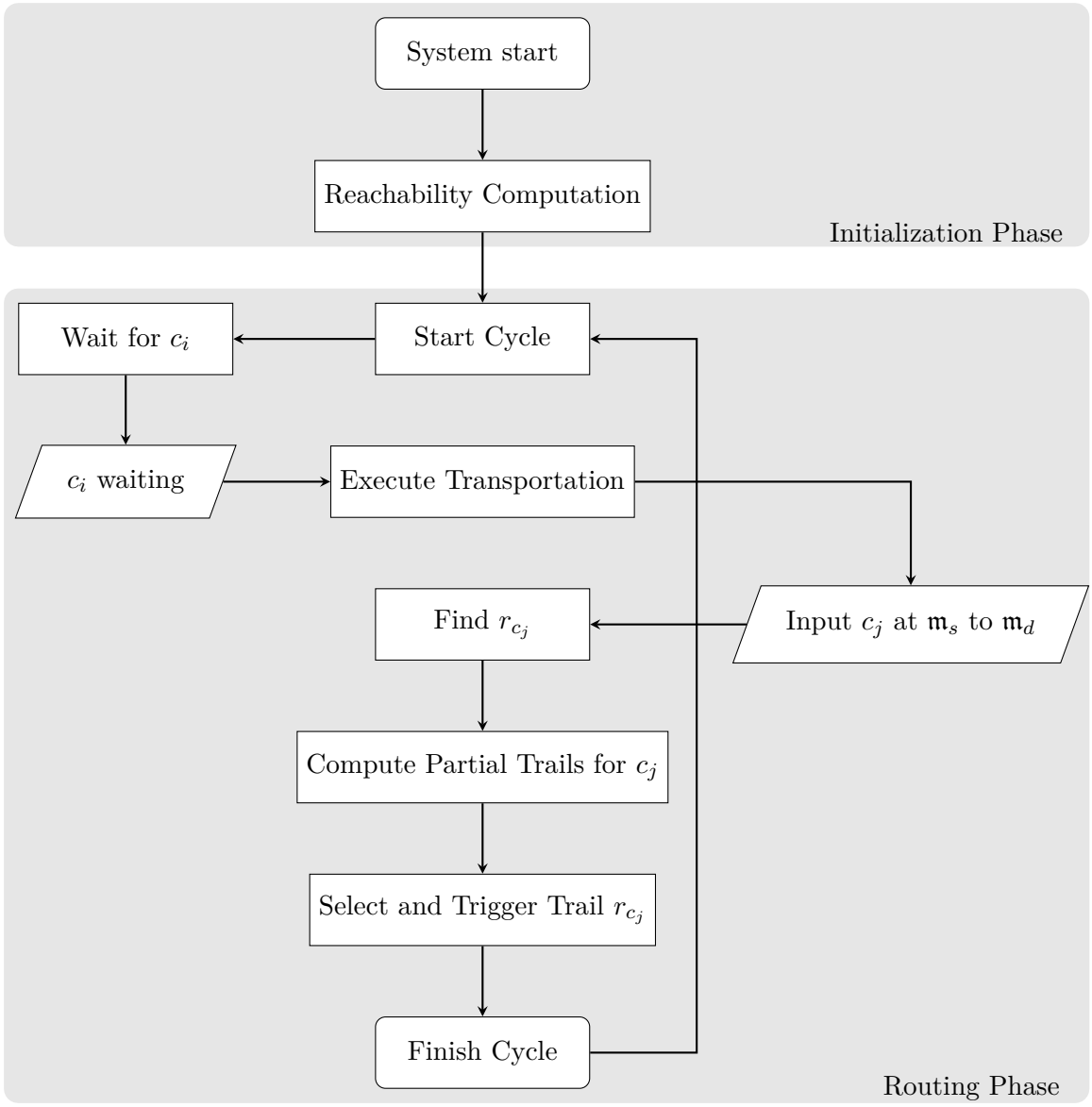


Figure 5.9: Processes from one $\mathfrak{m}_i$ to compute and execute a trail t_c.

Each route consists of one trail r_c which is computed during the transportation process such that changes in the system state[231] can be reacted to.

Initial static reachability determination

To ensure that the candidate set only includes feasible candidates that are able to reach the destination, a reachability method is used to create the initial routing table R_i for

[231] Like overloaded paths

each TU $\mathfrak{m}_i$ in the system. Furthermore, based on the routing table, the approximated time to the destination can be estimated prior to the transportation based on R_i.

Each $\mathfrak{m}_i$ can be marked as an interface at the system boundary. This means that $\mathfrak{m}_i$ can be a source, target, or transportation unit at a specific time τ. During one time period $[\tau, T], \mathfrak{m}_i$ can only enable the operation for one type, such that $\mathfrak{m}_i$ can not be a source- and target unit at the same time. In addition, a source, or target can hold more than one incoming or outgoing edge defined in a snapshot G_τ. If $\mathfrak{m}_i$ becomes a source , the current snapshot G_τ has to have at least one outgoing edge e_i. In contrast, if $\mathfrak{m}_i$ is a target , then G_τ it has to have at least one incoming edge e_j. By switching from a source to a target at G_τ to $G_{\tau+1}$, at least one edge has to be inverted such that $e_j = e_i^{-1}, e_i \in G_\tau \rightarrow e_j \in G_{\tau+1}$. Furthermore, a transportation unit has to have at least one incoming and one outgoing edge during one period $[\tau, T]$ at G_τ.

A cumulative, parallelized broadcasting method is used to compute the routing table on every $\mathfrak{m}_i$ storing the initial estimated transportation time to every possible target $\mathfrak{m}_j$. This method is in the class of distance vector routing by updating the current system status on TUs during run-time. Instead of all TUs, only selected TUs are updated to reduce the communication overhead. Distance vector routing is more light-weight than link state routing[232] because it does not consider and hold the whole network state, which is unnecessary for the used routing method explained later in section 5.3.2 for route selection. The introduced method stores only the time to the destination without the path and resolves the count to infinity error[233] compared to distance vector routing. The routing table $R_{\mathfrak{m}_i}$ at $\mathfrak{m}_i$ holds only one entry for each target unit $\mathfrak{m}_i$ and is updated during run-time. Instead of updating all $\mathfrak{m}_j$ in the system, only $\mathfrak{m}_j \in s \in r_c$ is updated to reduce the communication overhead. To compute the routing table at one $\mathfrak{m}_i$ to a destination $\mathfrak{m}_j$, a specific message structure is used to synchronize the times at which the method operates on.

The message to determine the times to one destination is a 2-tuple $(\mathfrak{m}_d, \tau)$ and stored in the routing table $R_{\mathfrak{m}_i}$ for $\mathfrak{m}_i$. The first element $\mathfrak{m}_d$ is the destination and the second element τ the cumulative time to the destination. To compute the routing table for one $\mathfrak{m}_i$ to a destination $\mathfrak{m}_d$ the decentralized algorithm 3 is used. Computing the routing table is triggered by a destination $\mathfrak{m}_d$ on startup itself. It starts the message with the

[232] Link state routing holds the whole system statuses at one node, which is not feasible for storing at PLCs due to the limitation in storage size and static array definition (see section 2.1).

[233] Ending in an endless loop while sending and broadcasting.

parameters $(\mathfrak{m}_d, 0)$ on every neighbor TU in N with source $\mathfrak{m}_d$. The triggered source $\mathfrak{m}_d$ is needed to compute the costs s_{ijk} and to ensure a loop-free message parsing.[234] Each TU $\mathfrak{m}_i$ computes its routing table $R_{\mathfrak{m}_i}$on its own, which enables a parallel computation in decentrally controlled TUs . First (line 2), the routing table $R_{\mathfrak{m}_i}$ is checked to see whether an entry for destination $\mathfrak{m}_d$ already exists. If not (line 3), an entry is created, with the cumulative costs τ based on the costs of the previous TU $\mathfrak{m}_s$. Thus, τ represents the costs from the destination $\mathfrak{m}_d$ to the last TU $\mathfrak{m}_s$. By adding the costs to the source $\mathfrak{m}_s$ the current operation costs are included in the routing table $R_{\mathfrak{m}_i}$ to reach the next element $\mathfrak{m}_s$ during transportation. This enables a distinction between different reachable TUs with heterogeneous costs. To overcome the excessive use of one specific $\mathfrak{m}_i$, the source $\mathfrak{m}_s$ is not linked in the routing table. This forces a computation during runtime which updates costs, such that τ can be interpreted as an indicator during run-time for reachability of the destination $\mathfrak{m}_d$. The start value of τ based on the computation in the initial phase represents the costs in the best-case scenario.[235] If an entry in $R_{\mathfrak{m}_i}$ for $\mathfrak{m}_d$ exists (line 5), it is replaced (line 8) with the new costs if the costs are smaller. Otherwise, the execution is terminated (line 6) because a shorter path has already been found. Afterwards, the set N with the local neighbors except the source $\mathfrak{m}_s$ is determined, and each element $\mathfrak{m}_j \in N$ receives a duplicate of the message with the new costs to compute its routing table $R_{\mathfrak{m}_j}$. The source $\mathfrak{m}_s$ is excluded in N to avoid recalculation, which can only result in higher costs [236].

Message structure for path determination

A well-defined message structure is needed to find several trails r_c for a container c. These trails are stored in the candidate set, evaluated, and the most promising trail r_c for c is selected for routing The aim of the message structure is to reduce the overall communication overhead in the network and ensure a loop-free trail finding. Therefore, a message m in the feasible candidate set F is a quadruple $(r_c, \mathfrak{m}_d, \tau, \epsilon)$ with:

- r_c as the current trail associated to the candidate; r_c is expanded during the graph traversal.

[234] This ensures a cancellation with negative edge costs of the algorithm and reduces the number of created messages in the system.

[235] There exists no blocking and deadlock, such that a flawless transportation exists.

[236] A transportation forth and backward on two elements makes no sense in continuous transportation.

Algorithm 3 Compute costs at TU $\mathfrak{m}_k$ to destination $\mathfrak{m}_j$ and add to $R_{\mathfrak{m}_k}$

Require: $\mathfrak{m}_i, \mathfrak{m}_s, c$ ▷ Current TU and computation trigger
Input: $(\mathfrak{m}_d, \tau)$ ▷ Possible TU destination and cumulative costs.

function ReachableDestination($\mathfrak{m}_d, \tau$)
  if $\nexists \mathfrak{m}_d$ **then**
    $R_{\mathfrak{m}_i} = R_{\mathfrak{m}_i} \cap \{(\mathfrak{m}_d, \tau + s_{c,\mathfrak{m}_d,\mathfrak{m}_i})\}$
  else if $\exists\,(\mathfrak{m}_j, \alpha) \in R_{\mathfrak{m}_i} : (\mathfrak{m}_i = \mathfrak{m}_j \wedge \alpha \geq \tau)$ **then**
    return ▷ A Shorter path already exists in $R_{\mathfrak{m}_i}$
  else
    $R_{\mathfrak{m}_i} = (R_{\mathfrak{m}_i} \setminus \{(\mathfrak{m}_d, \tau)\}) \cap \{(\mathfrak{m}_d, \tau + s_{c,\mathfrak{m}_d,\mathfrak{m}_i})\}$
  end if
  $N = \{v_i \mid v_i \in V_u, e_i \in E_u,$
  $(\text{head}(e_i) = v_i \wedge \text{tail}(e_i) = A^{-1}(\mathfrak{m}_k))$
  $\oplus\,(\text{tail}(e_i) = v_i \wedge \text{head}(e_i) = A^{-1}(\mathfrak{m}_j))\}$
  for all $\{\mathfrak{m}_i = A(v_i) \mid v_i \in N \wedge v_i \neq A^{-1}(\mathfrak{m}_s)\}$ **do**
    m_i. ReachableDestination($\mathfrak{m}_d, \tau + s_{c,\mathfrak{m}_d,\mathfrak{m}_i}$)
  end for
end function

- $\mathfrak{m}_d$ as the destination of c.
- τ as the costs in time for the trail r_c.
- ϵ is the maximal length of r_c for a candidate.

During exploration, $|r_c|$ increases and therefore τ does. Furthermore, when two or more messages $m_i, \ldots, m_j$ arrive at one TU they can be evaluated earlier by comparing their times τ. The message with the lowest τ is assessed further, while the other messages are dropped. If two or more messages have the same value τ, the message that arrived first is chosen.[237] During exploration, ϵ is decremented by 1 until it reaches 0. Afterwards, the evaluation of the remaining trails can begin.

Sequential candidate set computation

A decentralized method is used to compute the candidate set F with feasible trails r_c; its processes are visualized in fig. 5.10. First, the message m to compute F is initialized and multi-casted to all direct neighbors. Afterwards, a reachability check is conducted

[237] It is assumed that this m_i is able to react faster, because it was able to answer faster than the other messages.

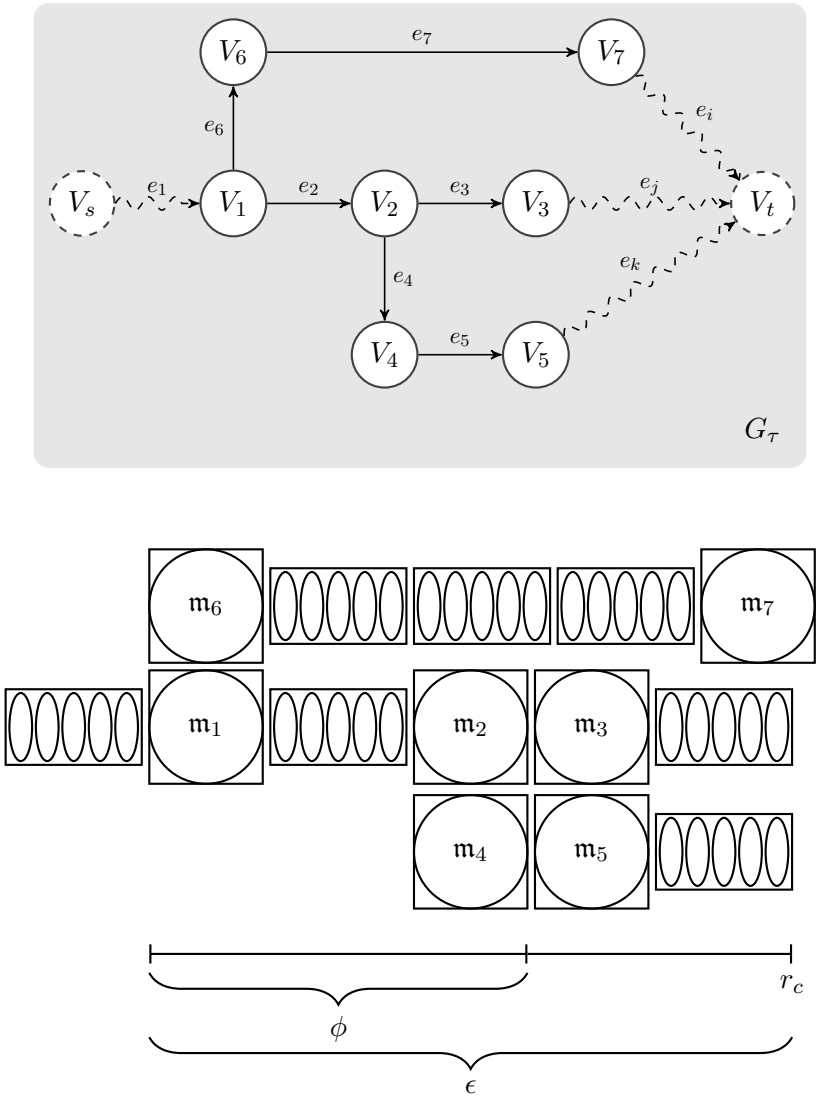


Figure 5.10: Concept to compute trail r_c.

based on $R_{\mathfrak{m}_i}$. Each promising neighbor receives the updated message m and adds its information to compute r_c. This updating and multi-casting of m to promising neighbors are done until the maximum length of r_c has been reached, or the destination has been found. Then, m is sent back to the sender and validated each time with local system state information until it reaches the source $\mathfrak{m}_s$. The source $\mathfrak{m}_s$ evaluates all r_c computed by $\mathfrak{m}_i$ and chooses the most promising one. Finally, the routing of c begins based on the chosen r_c.

The multi-cast method is used to determine trails and to drop dead-ends or unpromising trails beforehand. The message structure m is used and multi-casted to every direct

neighbor except the sender[238] of m to compute F. A neighborhood selection for multi-casting and a premature withdrawal method are used to reduce $|F|$. The same multi-cast method, as in algorithm 3 line 10, for N is used. Every neighbor receives m except the previous source of m. When receiving m for r_c to destination $\mathfrak{m}_d$, the premature withdrawal method drops the message when at least one of the following circumstances is given: $\mathfrak{m}_i \in s \in r_c \vee \mathfrak{m}_d \notin R_{\mathfrak{m}_i} \vee \epsilon \leq 0 \vee \mathfrak{m}_i = \mathfrak{m}_d$ (see algorithm 4, lines 2 and 4). The first term $\mathfrak{m}_i \in s \in r_c$ ensure a loop-free computation of r_c, because if the current $\mathfrak{m}_i$ is already in one $s \in r_c$ the computation is withdrawn, since $\mathfrak{m}_i$ has already been visited. Second, $\mathfrak{m}_d \notin R_{\mathfrak{m}_i}$ means the destination $\mathfrak{m}_d$ for c is not reachable by $\mathfrak{m}_i$. When $\epsilon \leq 0$, the maximum length for r_c has been reached and the currently computed r_c is sent back to the sender of m. If the last term $\mathfrak{m}_i = \mathfrak{m}_d$ holds the final destination for c is found, nd no further computation is needed.

Algorithm 4 Compute r_c at $\mathfrak{m}_i$

Require: $\mathfrak{m}_i, \mathfrak{m}_s$ ▷ Current and source TU

1: **function** ComputeTrail($(r_c, \mathfrak{m}_d, \epsilon)$)
2: **if** $\mathfrak{m}_i \in s \in r_c \vee \mathfrak{m}_d \notin R_{\mathfrak{m}_i}$ **then**
3: **return** ▷ Withdraw further computation
4: **else if** $\epsilon \leq 0 \vee \mathfrak{m}_i = \mathfrak{m}_d$ **then**
5: $F = F \wedge r_c$
6: **return** $\mathfrak{m}_i$ EvaluateTrail (r_c) ▷ See algo. 5 for evaluation of r_c
7: **end if**
8: **for all** $\mathfrak{m}_j \in N_{\mathfrak{m}_i}$ **do**
9: $t = \text{FindTimeSlot}(r_c, \mathfrak{m}_j, \text{cost}\,(r_c))$
10: $s = ((\mathfrak{m}_i, \mathfrak{m}_j), t_1, t_2)$ ▷ t_1 as release date and t_2 as due date
11: $r_c = r_c \cap s$
12: $\mathfrak{m}_j$. ComputeTrail $(r_c, \mathfrak{m}_d, \epsilon - 1)$
13: **end for**
14: **end function**

Algorithm 4 is the decentralized method to compute F. During every multi-cast, the message m is duplicated and the new duplicates are treated as a new trail. Every new trail r_c distinguishes itself from the previous one by a newly added TU $\mathfrak{m}_j$ (line 11).

[238] Source and sender are differentiated. A Source $\mathfrak{m}_s$ is the origin of m, while a sender $\mathfrak{m}_j$ is the previous TU multi-casting m to the current $\mathfrak{m}_i$. In the beginning, $\mathfrak{m}_s = \mathfrak{m}_j$ holds and during computation $\mathfrak{m}_s \in s_1 \in r_c$.

5.3.2 Path selection

After the decentralized computation of F a metric is needed to select the most promising candidate $r_c \in F$ as a route for c. The goal is to minimize the transportation time of c and causes as little blocking with other containers in the system as possible. Therefore, each $\mathfrak{m}_i$ in the system receiving F selects one promising r_c and forwards its candidate back to the source. This reduces the amount of $|F|$ and the computation time to compute one r_c, because during each forwarding a set of trails is further removed. Moreover, due to communication insecurity[239] an assumption is made regarding the waiting time: if an $\mathfrak{m}_i$ can not respond in a certain period of time, it is assumed to be off-line.

Decentralized promising candidate selection

As previously mentioned, each $\mathfrak{m}_i$ recursively selects one candidate of the received candidate set F to the source, which selects the candidate r_c to use for routing. It is assumed that this last candidate was able to win against the other candidates in $F_{\mathfrak{m}_i}$. Despite a global F, each $\mathfrak{m}_i$ holds a local set $F_{\mathfrak{m}_i}$ with the feasible local trails computed by its children in the communication graph M_c. During iteration, the parent merges the local sets $F_{\mathfrak{m}_i} \dots F_{\mathfrak{m}_j}$ of its children by selecting the most promising trail r_c in each set. The newly merged set holds then all promising trails r_c at $\mathfrak{m}_i$. This recursive merging is done up to the source of route computation, which selects the trail r_c for execution.

Algorithm 5 shows the method running on each TU $\mathfrak{m}_i$ respectively. Each $\mathfrak{m}_i$ goes into a waiting mode (line 2), waiting for selected trails r_c. It uses a dynamic waiting time computation which differs depending on the depth of $\mathfrak{m}_i$ in M. The most promising local trail r_i (line 3) is used. It is compared to each receiving trail r_c (line 6) during the waiting time. When $\text{cost}(r_i) > \text{cost}(r_c)$ the new receiving trail r_c is selected and assumed to be dominant[240] over r_i. When the waiting time is over (line 9), the most promising trail r_i is sent to the parent (line 11) in M for further evaluation. If $\mathfrak{m}_i$ is the root in M, the last remaining trail r_i is used for routing c because it dominated in the evaluation. Therefore, it is assumed that r_i is the route with the lowest cost.

[239] It can not be assumed that a $\mathfrak{m}_i$ answers in a proper time, e.g. the global waiting time for an answer has been exceeded. Due to failure or exhausting computation cause by other operations $\mathfrak{o}_i$ on $\mathfrak{m}_i$.

[240] Faster in transportation time.

The auxiliary functions HasParent($\mathfrak{m}_i, M_c$) and GetParent($\mathfrak{m}_i, M_c$) are defined as:

$$\text{HasParent}(\mathfrak{m}_i, M_c) = \begin{cases} 0 & \text{if } \mathfrak{m}_i \text{ is root in } M_c \\ 1 & \text{else } \mathfrak{m}_i \text{ has parent } M_c \end{cases} \tag{5.22}$$

$$\text{GetParent}(\mathfrak{m}_i, M_c) = \begin{cases} 0 & \text{if } \mathfrak{m}_i \text{ is root in } M_c \\ 1 & \text{else } \mathfrak{m}_i \text{ has parent } M_c \end{cases} \tag{5.23}$$

Both functions need the communication graph M_c as a second parameter because searching for a trail for n containers generates n communication graphs. This enables a parallel search for trails.

Algorithm 5 Evaluate F at $\mathfrak{m}_i$

Require: $\mathfrak{m}_i$ ▷ $\mathfrak{m}_i$ as the current TU.

```
1: function EvaluateTrail(r_c)
2:     if NotWaitingForEvaluation then
3:         r_i = r_c                                   ▷ Most promising trail.
4:         δ = τ + EvalWaitingTime()                   ▷ See algo. 5.24
5:         WaitForEvaluation(δ)
6:     else if cost(r_i) > cost(r_c) then
7:         r_i = r_c
8:     end if
9:     if δ > τ then                                   ▷ τ as the current system time.
10:        if HasParent(m_i, M_c) then
11:            return GetParent(m_i, M_c).EvaluateTrail(r_i)
12:        else
13:            ReservateRoute(r_c)                     ▷ See algo. 6
14:        end if
15:    end if
16: end function
```

Waiting time for computing most promising trail

In decentralized transportation systems, communication between TUs is crucial. It can not be assumed that the requested data will arrive on time or worse, at all. Furthermore, in highly complex systems, one TU does not know the size of the system and is therefore not able to estimate a waiting time for a response. Especially with the introduced method of searching for new trails with multi-casting, the maximum waiting time differs

based on the found trails and their current communication workload, which can cause a delay in responding to a request. The dynamic waiting time for promising trails uses a similar technique as the waiting time at one $\mathfrak{m}_i$ in section 5.2.1 and is restricted by the communication waiting and response time defined in section 5.1.3. The used waiting time is computed by evaluating historical data and changes during run-time while new data is added to the set of historical data at one $\mathfrak{m}_i$.

The waiting time ω describes the maximum waiting time for a response using the search method for new trails r_c for a container c. In the beginning $\omega = 30s$ based on the usual default waiting time in network communication. During each trail search, the real waiting time based on the remaining maximal path length ϵ is stored in the set H_ϵ at each $\mathfrak{m}_i$. The differentiation between the remaining ϵ is crucial due to the expected waiting time. It can be assumed that a lower ϵ will respond faster because fewer TUs are considered compared to a higher ϵ.

$$\text{EvalWaitingTime} = \begin{cases} 30 & \text{if } H = \emptyset \\ \frac{1}{2}\left(H_{n*p} + H_{n*p+1}\right)) & \text{if } |H| \mod 2 = 0 \\ H_{\lceil n*p \rceil} & \text{else} \end{cases} \tag{5.24}$$

Eq. 5.24 shows the computation of the maximum waiting time for trails during evaluation. The auxiliary variable $n = |H|$ is used to compute the p-quantile. Choosing p in the preprocessing directly affects F and its trails. A lower p causes a withdrawal of feasible trails r_c not responding in time, which could be caused by overloaded communication interfaces at one $\mathfrak{m}_i$.[241]. If the chosen p is too high, trails are considered that are valid but not feasible. This causes unnecessary computation and waiting time at one $\mathfrak{m}_i$ by evaluating unfeasible trails. Therefore, p has to be chosen such that feasible trails are considered while unfeasible trails are withdrawn, thus reducing the computation and waiting time.

Path cost identification and path selection

Finally, the trail r_c has to be selected and marked for reservation for one container c. As already explained in section 5.3.2 with algorithm 5, the candidate is selected by using

[241] For example multiple search for different trails or routing synchronization explained in section 5.3.3

the remaining trails in F at the source $\mathfrak{m}_i$ of m for trail search. This is done by choosing r_c with the lowest costs described by the following equation:

$$\min \operatorname{cost}(r_c), r_c \in F \tag{5.25}$$

Afterwards, every $\mathfrak{m}_i \in s \in r_c$ is notified and s is transformed into a fixed job in the scheduler at every $\mathfrak{m}_i$, which is called route reservation.

Route reservation

After a trail r_c is selected for routing by the source $\mathfrak{m}_s$ which triggered the decentralized search method, each $\mathfrak{m}_i \in s \in r_c$ has to be notified. Every $\mathfrak{m}_i$ related to V_i from the source $\mathfrak{m}_s$ to the destination $\mathfrak{m}_d$ is notified. During the notification, each $\mathfrak{m}_i$ transforms local trail information into a job for the local scheduler. After the transformation, $\mathfrak{m}_i$ is capable of routing c with local information with minimum communication effort.

Algorithm 6 Evaluate F at $\mathfrak{m}_i$

Require: $\mathfrak{m}_i, \phi$

```
1:  function ReservateRoute(r_c)
2:      if A^{-1}(m_i) ∈ V_u then
3:          j = Split(r_c, m_i)                          ▷ See algo. 1
4:          S_{m_i} = S_{m_i} ∩ {j}
5:          phi − −
6:          if φ = 0 then
7:              m_i. TriggerPartialRouteComputation
8:          end if
9:      end if
10:     if m_i = m_d then
11:         return
12:     end if
13:     m_{i+1}. ReservateRoute(r_c)        ▷ m_{i+1} as the subsequent TU of m_i
14: end function
```

Algorithm 6 shows the sequential job reservation after a trail r_c is chosen by $\mathfrak{m}_s$ from $\mathfrak{m}_s$ to $\mathfrak{m}_d$. Each $\mathfrak{m}_i$ including $\mathfrak{m}_s$ and $\mathfrak{m}_d$ executes algorithm 6 in chronological order given by r_c. In line 2, a check is made to see whether the current $\mathfrak{m}_i$ is related to a $V_i \in V_u$ to reduce the number of schedules in the system. Thus, $A^{-1}(\mathfrak{m}_i) \in V_u$ is only given if $\mathfrak{m}_i$

is connected to at least three more TUs (see section 5.1.1). Afterwards, a job j for the scheduler $S_{\mathfrak{m}_i}$ at $\mathfrak{m}_i$ is created (line 3) and stored (line 4). Storing j finally blocks the time window in $S_{\mathfrak{m}_i}$ and can cause a new snapshot $G_{\tau+1}$ (see fig. 5.10), which affects future searches for trails. If $\mathfrak{m}_d$ is reached (line 6), the route reservation for r_c finishes and all jobs for each $\mathfrak{m}_i$ have been created. Finally (line 9), the successor $\mathfrak{m}_{i+1}$ receives the message with the same parameters to compute its job to block the time window to transport the container on arrival.

Furthermore, to begin a new partial route computation before reaching the end of r_c, the attribute ϕ is used. It defines the unit $\mathfrak{m}_i$ to calculate a new route from the destination $\mathfrak{d} \in r_c$ as a starting point for future routes to reach the final destination of container c.

After all three steps (candidate set computation, candidate selection, and r_c reservation), the preprocessing phase for transporting c ends and the system is in a stable state and can provide the necessary information to take action[242] to transport c.

5.3.3 Route execution

After a trail r_c for container c is found, and every TU $\mathfrak{m}_i$ transforms each $s \in r_c$ into a job $\mathfrak{j}_{\mathfrak{m}_i}$, the routing for c can be executed. The information concerning c is communicated sequentially during transportation, which is triggered by a sensor logic implemented at a low control level.[243]. Simple route executions are folded into edges in G_τ and are controlled by $A(V_i) = \mathfrak{m}_i$ by passing c to them. Therefore, the direction is given by the handover of c from $\mathfrak{m}_i$ to the first $\mathfrak{m}_j \in \text{Unfold}\,(e_i)$.

Several cases can occur for one container during transportation:

- Arrives on time: When c arrives at $\mathfrak{m}_i$, on time and the path to the next TU is empty, c can be transported to the next $\mathfrak{m}_j$.
- Arrives too early: The path between $\mathfrak{m}_i$ and $\mathfrak{m}_j$ is checked, if it is free then c s transported.

[242] E.g. information to control the motors in a feasible way such that the container is moved in the right direction to the destination.

[243] PLC control level. The triggering of sensors is caused by c moving over them. Currently, simple binary photoelectronic sensors are used as exemplary reference. When c passes, a binary signal is sent to the PLC, which is capable of computing whether c enters or leaves the $\mathfrak{m}_i$.

- Arrives too late: The same steps occur as when the container arrives too early; c is transported when the route is free.

If only one container exists for transportation at every $\mathfrak{m}_i \in s \in r_c$ it can be forwarded easily. Therefore, these cases are trivial.

When other containers have to be considered, the system stability has to be ensured. This is done using an online decentralized deadlock check to fetch others $\mathfrak{m}_j$ statuses for coordination at $\mathfrak{m}_i$ and storing them in the set $B_{\mathfrak{j}_i}$ with $\mathfrak{j}_i$ related to one c at $\mathfrak{m}_i$ as $\mathfrak{j}_{c,\mathfrak{m}_i}$. Its concept is to find online and in real-time inverse flows by searching for and analyzing other flows based on local information. Afterwards, synchronization is done with $\mathfrak{m}_j$, that may cause deadlocks. Finally, a deadlock resolving method enables future secure transportation.

Congestion determination at container arrival

An inverse path finding procedure is used to ensure local deadlock-free routing between two TUs during transportation. When a scheduled container c follows a trail r_c and is about to arrive at $\mathfrak{m}_i, A(V_i) = \mathfrak{m}_i$ with an edge from $e_i = (V_i, V_j), A(V_j) = \mathfrak{m}_j$, a deadlock check is conducted for e_i using a handshake between $\mathfrak{m}_i$ and $\mathfrak{m}_j$. The handshake procedure blocks e_i in G_τ for a time α such that G_ϵ cannot change from e_i to e_i^{-1} until $\epsilon > \alpha$.

Before actions to avoid deadlock can be taken, the set $B_{\mathfrak{j}_i}$ must be determined. If $B_{\mathfrak{j}_i} = \emptyset$, the container c can be forwarded to the next $\mathfrak{m}_j \in s \in r_c$. Otherwise, c is blocked in front of $\mathfrak{m}_i$ at $\mathfrak{m}_k$ such that every c with $e_i = (V_i, V_j), V_j \neq V_k$ transportation can be executed. In summary, $B_{\mathfrak{j}_i}$ is related to one job $\mathfrak{j}_i$ at $\mathfrak{m}_i$ to block the execution of $\mathfrak{j}_i$ at $\mathfrak{m}_i$ for c. The first step is to recognize whether a handshake must be made between two $\mathfrak{m}_i$ and $\mathfrak{m}_j$. This is done using a linear inverse flow search on the local scheduler at $\mathfrak{m}_i$ by evaluating each job $\mathfrak{j}_{c,\mathfrak{m}_i}$ before the arrival of c.

Algorithm 7 shows the computation of $B_{\mathfrak{j}_i}$. This is done right before the arrival of c at $\mathfrak{m}_i$, such that if an inverse flow exists it can be resolved by $\mathfrak{m}_i$. Afterwards, once no inverse flow exists, c is accepted for handling and the waiting time ends for the previous $\mathfrak{m}_k$ blocked by $\mathfrak{m}_i$ with c. To compute $B_{\mathfrak{j}_i}$, an inverse flow e_i^{-1} is searched for comparing every edge e_j to e_i (line 2). When $e_j = e_i^{-1}$ (line 3), an inverse flow exists; a check is

Algorithm 7 LinearInverseFlowSearch for $\mathfrak{J}_{c,\mathfrak{m}_i}$ at $\mathfrak{m}_i$

Require: $\mathfrak{m}_i$

```
1: function LinearInverseFlowSearch(j_{c,m_i})
2:     for all j ∈ J_{m_i} do
3:         if e_i^{-1} = e_j, e_i ∈ j_{c,m_i} ∧ e_j ∈ j then
4:             r_part := r_c \ {s_1, ..., (m_i, α, β)}
5:             r_part = r_part \ {m_{j+1}, ..., m_n}
6:             if τ + cost(r_part) ≥ ρ_{c,m_j} then
7:                 B_{j_i} = B_{j_i} ∩ {e_j}
8:             end if
9:         end if
10:    end for
11:    return B_{j_i}
12: end function
```

conducted to see if e_j and e_i^{-1} interfere with other in time (line 4 and 6). If an interference occurs (line 6), the edge e_j is added to the set $B_{\mathfrak{j}_i}$ (line 7) for further resolution. The used auxiliary variable r_{part} stores the route between $\mathfrak{m}_i$ and $\mathfrak{m}_j$, and is used to compute the time needed for this part of the route to check whether a deadlock may occur.

Inverse flow handling and resolving

To avoid deadlocks, a handshake method is used. At each arrival of a container c at $\mathfrak{m}_i$, the inverse flow is computed by algorithm 7 and the edges $e_i, \ldots, e_j$ from to vertices V_i, V_j are blocked until c arrives at $\mathfrak{m}_j$. This is done as follows.

First, when c_i arrives at $\mathfrak{m}_i$ with e_i and c_2 arrives at $\mathfrak{m}_j$ with e_j such that $e_i = e_j^{-1}$, a decision must be made concerning container, c_i or c_j, must be handled first and transported between $\mathfrak{m}_i$ and $\mathfrak{m}_j$ the direction of e_i or e_j. This is done in a first-come, first-server manner.[244] The first arriving container c_i will trigger a message from $\mathfrak{m}_i$ to check for inverse flows at $\mathfrak{J}_{\mathfrak{m}_i}$ and find the flow e_j for c_j with algorithm 7. Then, $\mathfrak{m}_i$ notifies $\mathfrak{m}_j$ and stores all containers c in $H_{\mathfrak{j}_{c,\mathfrak{m}_j}}$.[245] $H_{\mathfrak{j}_{c,\mathfrak{m}_j}}$ represents the set of containers holding c_j from further executing on $\mathfrak{m}_j$. Afterwards, $\mathfrak{m}_j$ adds c_i to $H_{\mathfrak{j}_{c,\mathfrak{m}_j}}$ on arrival of the notification from $\mathfrak{m}_i$ and $\mathfrak{m}_j$. $\mathfrak{m}_j$ replies with an acceptance message. When the acceptance

[244] If two messages occur simultaneously, the message that arrives first at the destination will be chosen. Thus, it depends on the underlying communication structure.

[245] The difference between the sets B and H is that B holds containers that could cause a possible deadlock, while H stores the containers that will cause a deadlock if transported any further.

message arrives at $\mathfrak{m}_i$ and $H_{\mathfrak{j}_{c_i,\mathfrak{m}_i}} = \emptyset$ is given c_i transportation is executed based on r_{c_i}. Finally, when c_i arrives at $\mathfrak{m}_j$, c_i is removed from $H_{\mathfrak{j}_{c_j,\mathfrak{m}_j}}$ and c_j transportation can be executed.

Blocking can occur between n containers in the system. As long as a part of a trail r_{part} is given and no time is violated, a container c can move when its current holding list $H_{\mathfrak{j}_{c_i,\mathfrak{m}_i}} = \emptyset$.

6 Implementation

This chapter discusses the implementation of the introduced method for routing containers in a decentralized complex transportation system presented in section 5. As a basis, the method and the environment are being implemented in Unity3D.[246] It provides the capability in decentralized computation, which is used as a basis to represent decentrally structured transportation units. As an advantage, a visual evaluation is also possible by mapping the abstract structure of graphs to visual representation, e.g. conveyors and containers.

6.1 Decentralized complex System implementation

This section provides an overview of the implementation of TUs $\mathfrak{m} \in \mathfrak{M}$, the moving container $c \in C$ and the environment on field level, as well as a WMS Client controlling the arrival of c at a time $\tau \in T$. This includes the field level implementation by simulating motors and sensors and the communication between TUs at communication level. This enables local control and global control of c by moving it from a source $\mathfrak{m}_s$ to a destination $\mathfrak{m}_d$.

Intelligent device representation

As a basis, three main TUs have been implemented which can switch roles depending on evaluation parameters. Based on these units, the containers are transported from s to d during run-time. Each TU is equipped with one intelligent device controlling its behavior.

Each component has the following similarities:

[246] www.unity3d.com

- Max speed: A triple-vector defining the transportation speed (x, y, z).
- Interfaces: Connection to other transportation units. Defines a direct communication to other units, which can be a wireless or a wired connection.
- Initial costs: The first cost value assumed for the dynamic cost model.
- ϕ: Threshold before a new path is computed.
- ϵ: The maximum size of a trail r_c.
- Scheduler: Local scheduler storing all containers and their arrival, destination, and next hop.

The first TU is a unit with one connection only; it switches roles between a source and a sink. In industrial application, it is an interface between automatic systems and its environment. During a time τ, a unit $\mathfrak{m} \in \mathfrak{M}$ can switch roles, but $\mathfrak{m}$ can only have one role during a time-slot $[\tau, \tau + 1]$. In addition to the previous parameters, this component, arrival rate, when a source, can be set. Like a sink, no further attributes can be set. Its main task is to remove the container from the simulation environment. The second TU is a conveyor for continuous transportation with the capability of bidirectional transportation. It consists of one motor and two sensors. The motor represents the forces transmitted to the rollers for moving the containers in one direction. One sensor at each end is used to determine the arrival and departure of one container. The arrival or departure is derived by the motor status and the sensors. When the motor moves the container to the left, and the left sensor determines a rising edge followed by a falling edge, it can be assumed that the container has left the TU and has been arrived at the next one. This TU‘s main task is the transportation of containers between the discontinuous units only. This unit is restricted to moving the container forwards, backwards, or stopping it. The third TU is a discontinuous TU enabling forking and joining within the transportation system. In contrast to the conveyor unit, this one can have more than two interfaces. Its main task is to computing r_c since it has control over distributing the container within the system.

Fig. 6.1 shows a small implemented system for evaluating the functionalities of TUs. The small squares in the middle are right angle decks able to connect several TUs and to form a path. Each is connected with a conveyor for continuous transportation of containers. At the left and right ends, TUs are placed that are able to switch between a

source and a sink. Routes can be formed from the left to the right, from right to left, or on one side only. A source cannot be a sink for one container.

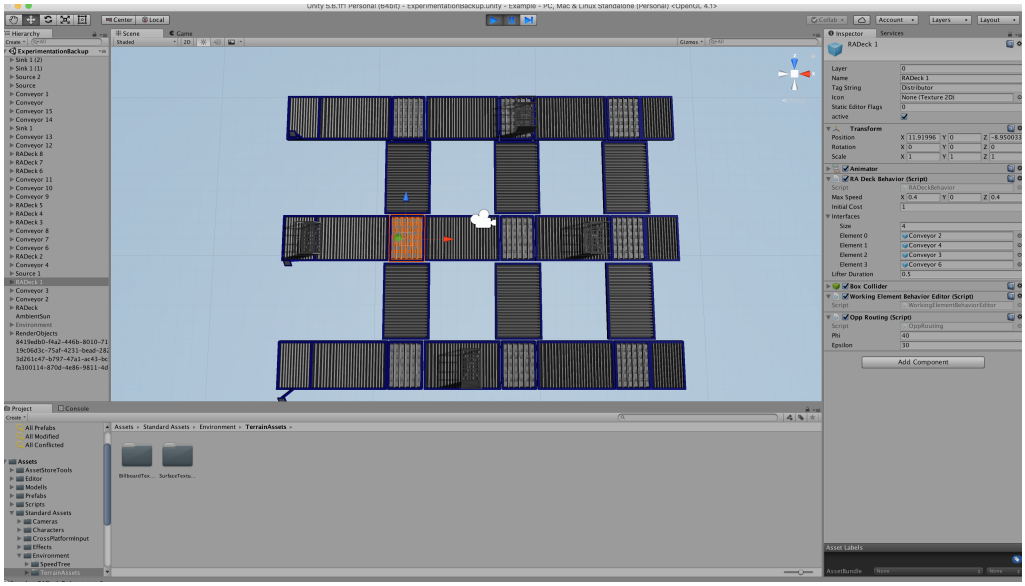


Figure 6.1: Implemented simulator for decentralized organized transportation systems.

Implementation of communication layer

Each TU is equipped with decentralized control logic. There exists no access to global state information. Furthermore, to provide time-invariant, there exists no global scheduler controlling the local time of each TU. To provide field-level control, each TU has a logic based on the restrictions given in industrial systems to intelligent devices, which are limitations in computational speed and memory (see section 2.1). In addition, every TU is equipped with a communication unit (in the simulation and a software layer). Since a control logic works in cycles, the communication also considers this with periodical communication.[247]

The messages are divided into route planning (described in the next section) and route execution. Messages of the route execution type have higher priority than planning, since they include local synchronization between TUs $\mathfrak{m}$ to enable a fluid transportation of a container $c \in C$. It is assumed in the implementation that c triggers the sensor of $\mathfrak{m}_i \in \mathfrak{M}$ with the front face and some seconds are still available until it reaches the next

[247] At the beginning of a cycle, the buffer/memory storing the arrived message is read and at the end of the cycle, the message to be sent is written to the buffer before being sent after the cycle ends.

$\mathfrak{m}_j \in \mathfrak{M}$. During this time, the synchronization between $\mathfrak{m}_i$ and $\mathfrak{m}_j$ is done. Every $\mathfrak{m}$ manages a blocked list in a first-in, first-out (FIFO) [248] order with all blocked C. Once an $\mathfrak{m}_j$ is free, it handles the next c_i in the blocked list. If a c from $\mathfrak{m}_i$ is being blocked by $\mathfrak{m}_j$, and receives a blocking answer, then it stops the moving of c until $\mathfrak{m}_j$ receives a free message to $\mathfrak{m}_i$. Then the movement of c starts and is handed over to $\mathfrak{m}_j$.

Every message additionally stores the path it has been transmitted, by adding itself to the path. When $\mathfrak{m}$ appears in the path, the message is dropped and removed from the system. This ensures a loop-free messaging.

6.2 Implementation of the opportunistic routing method

This section introduces the implementation of the identification and representation of the underlying graph G_u, the message structure for the planning of R_c, and the storing of the values for later transport execution.

Protocol structure for routing

To identify the underlying graph G_u, it is assumed that each $\mathfrak{m}_i$ is connected by a wired communication solution[249]. During the startup of the system, each $\mathfrak{m}_i$ checks its communication interfaces to see whether they are alive. If so, a message with information is pushed back to $\mathfrak{m}_i$ and will be stored. Otherwise, after a specified amount of time, $\mathfrak{m}_i$ assumes there is no connection at one interface. The next step is to transform every $\mathfrak{m}_i$ to a vertex V_i or an edge E_i depending on the number of communication interfaces for later route planning. The transformation by the number of interfaces is defined as follows:

- 1: The $\mathfrak{m}_i$ obtains the ability to be a source or sink, respectively of τ, if not specifically defined to be one type only. In the underlying graph, it will becomes an vertex V_i.

[248] The element that enters the list first is removed first.

[249] E.g. RJ45. Wireless solutions are possible as well but not investigated; if interested, see [PLM02], pp. 112-118, which introduces solutions capable of identifying G_u by a wireless connection.

- 2: Every connected $\mathfrak{m}_i, \ldots, \mathfrak{m}_j$ with two interfaces only will form an edge E_i between two vertices (V_i, V_j).

- > 2: These $\mathfrak{m}_i$ are able to join and fork the transportation flow. They are also able to define a route by determining a direction; therefore they will become a vertex V_i.

Based on the created G_u, the first snapshot in the dynamic network $G[0, T]$ at T_0 is defined as $G_0 = (V_0, E_0)$.

To enable a quick feasibility check if a specific direction given by V_i is able to reach a destination V_d a flag is being used indicate the reachability of V_d at each $v \in V$. This is initiated by every possible destination by flooding the network with a simple message storing the destination V_d.

MHE evaluation

Every $\mathfrak{m}_i \in \mathfrak{M}$ evaluates itself without global state knowledge. As explained in section 5.2.1, the static cost model is computed locally during one cycle. Based on the parameters defined previously for one $\mathfrak{m}$ the static cost model is derived upon the startup of the system. The dynamic part is determined by evaluating trails passing through. Every time, the necessary values described in section 5.2.2 are stored during one computational cycle.

Decentralized path finding

A route is determined based on G_u as explained in section 5.3. To enable this operation in a decentrally organized transportation system, a message handling independent of the cycles must be implemented. Therefore, sub-message types are used, defining one specific operation for one cycle at one $\mathfrak{m}_i$. These are the following:

- ORStartNodeFinding: Initializes all needed data structures as preparation for path finding.

- ORExploration: Used to find the destination or at least come close to the destination for future r_c finding.

- ORBack: Traverses back and spreads the collected information among the visited nodes.

A message time-out reflects the dynamics of complex networks that an $\mathfrak{m}_i$ can break-down. If it cannot be contacted for an adjustable amount of time, it is assumed that $\mathfrak{m}_i$ is not longer operable. On the other hand, if a new $\mathfrak{m}_i$ appears in the system, it will collect the information from connected $\mathfrak{m}_j \in \mathfrak{M}$, and afterwards adds it to G_τ. During run-time, the underlying graph G_u is of no use and is, therefore being removed from the memory. All operations are done on the complex graph G only.

7 Experimental design

This chapter shows the validation of the method proposed in chapter 5 and is based on the implementation described in section 6. It checks whether the protocol enables a proper system representation with considering changes over time whether the evaluation of transportation units is able to reflecting the current and future status of transportation units, and whether the routing method finds and executes routes without causing deadlocks.

The data which has been used as parameters for the validation are real planning transportation data. They were provided by an intralogistics supplier that has been anonymized such that no conclusion can be made. The usage of real planning data should give insight into the real capability of the method in an operative environment.

Finally, the last sections evaluate the proposed assumptions from section 2.3 respectively. Each element noted as a challenge is analyzed, and it is shown that the introduced method addresses each of these challenges.

7.1 System description

First, the section explains the parameters, which are based on real planning transportation data given by an intralogistics supplier. These parameters are globally used to validate each of the three sub-methods providing a proper system running by routing a set C of containers through the system. First, the section covers the parameters used for the simulation described in chapter 6 will be explained. This is done by providing first an overview of each unit type, and then of the system as a whole. Based on the parameters in the simulation, the evaluation of each of the three modules is then explained in consideration of its scalability, flexibility, and its robustness.

7.1.1 Transportation units

As proposed in section 5.2, the system consists of containers, which are moved between a predefined source and destination[250], at TUs, whereas simple bidirectional TUs and PEs for positioning a container c in the system for distribution, e.g. join or fork transportation lines.

In the evaluation, the system's environment consists of the ability to provide communication within each TU, to give information from the upper logic defined in WMS Client, and on the field level the simulation of the sensor and motor data for enabling movement of a container. The communication is being assumed to operate instantaneously and perfectly between two TUs such that no network errors, e.g. blocking, is considered. At the WMS Client level, the arrival of a container is parameterized at the source and follows a distribution. Sensor and motor data are simulated as being processed in one PLC. A separate memory collects and provides the information at the beginning of each process cycle of one PLC, and the motor parameters are written at the end of each cycle. Afterwards, when the parameters have been set, the motor reacts to these values, by running faster / slower, or changing directions.

Containers

A container $c \in C$ in the system is a construct which that is moved from a source $\mathfrak{m}_s$ to a destination $\mathfrak{m}_d = \mathfrak{d}$. Its arrival time τ, its source $\mathfrak{m}_s$, and destination $\mathfrak{d} \in \mathfrak{D}$ are transmitted by the upper logic operating in the WMS-Client organizing the material flow within the intralogistic system. Homogeneous containers are moved and have rectangle dimensions of $1.5m$ in length, width, and height. When generated at the source, it is assumed that the source is notified about the existence of c and is, therefore, able to start the process of finding a proper route r_c.

Bidirectional TUs

In this evaluation, bidirectional TUs are roller conveyors with two sensors following the

[250]This information and its arrival are provided by the upper logic processing in the WMS Client, as defined in the controls pyramid in section 2.1.

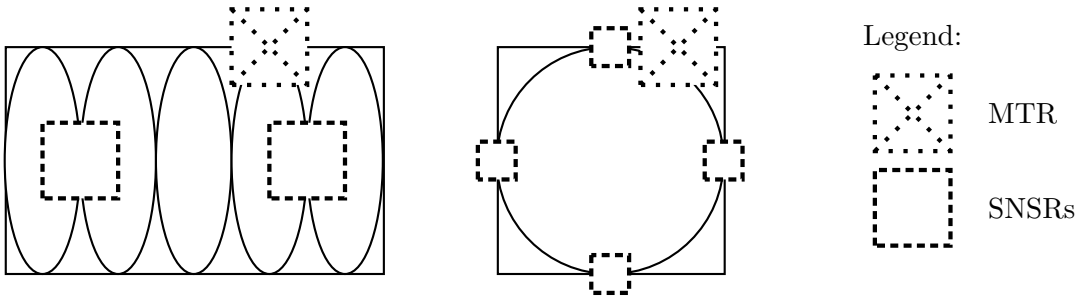


Figure 7.1: Exemplary picture of one transportation unit (TU, left) and one positioning equipment (PE, middle) including the placement of motors (MTR) and sensors (SNSR).

creep-speed paradigm[251] to control the movement of one or more $c \in C$. In the following sections, one TU can handle at most three containers at the same time. Because of the continuous movement, when one container $c_1 \in C$ is moved, a second $c_2 \in C$ and third $c_3 \in C$ are moved as well. This is due to the motor structure at one TU: it is assumed when turned on that all rollers move simultaneously, e.g. one belt connects all rollers with the motor. Two sensors (sensor block, see fig. 7.1 left side, SNSRs) are placed as a group in the front and two at the back of the TU, such that the entry of one c can be determined while the two sensors on the other side recognize the arrival at the end of the TU and are able to stop c. Every bidirectional TU $\mathfrak{m} \in \mathfrak{M}$ is parameterized as follows:

$$\text{Neighbors} : \mathfrak{N}_\mathfrak{m} := \{\mathfrak{m}_j, \mathfrak{m}_k\}, \mathfrak{m}_j \in \mathfrak{M}, \mathfrak{m}_k \in \mathfrak{M} \tag{7.1}$$

$$\text{Operations} : \mathfrak{O}_\mathfrak{m} := \{\text{move\_left}, \text{move\_right}\} \tag{7.2}$$

$$\text{Sequences} : \mathfrak{l}_1 = \{\mathfrak{o}_1\}, \mathfrak{l}_2 = \{\mathfrak{o}_2\} \tag{7.3}$$

$$\text{States} : \mathfrak{S}_\mathfrak{m} = \{Idle, WorkingLeft\ WorkingRight, Waiting\} \tag{7.4}$$

$$\text{Preconditions} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : \text{pre}(\mathfrak{o}) = \mathfrak{S}_1 = Idle \tag{7.5}$$

$$\text{Postconditions} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : \text{post}(\mathfrak{o}) = \mathfrak{S}_1 = Idle \tag{7.6}$$

$$\text{Velocity} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : v_\mathfrak{o} = 0.4ms \tag{7.7}$$

$$\text{Acceleration} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : a_\mathfrak{o} = 0.2ms^2 \tag{7.8}$$

$$\text{Distance} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : d_\mathfrak{o} = 4.5m \tag{7.9}$$

[251] The creep-speed paradigm describes the control of a c by slowly reducing its speed when arriving at the first sensor, e.g. 0.1m/s, and stopping c when arriving at the second sensors. Its advantage is that the position of c can be better controlled by invoking a slip by slowly reducing the speed instead of suddenly stopping.

$$\text{StaticProcessingTime} : \phi_{\mathfrak{m},\mathfrak{l}} = \frac{d_{\mathfrak{o}}}{v_{\mathfrak{o}}} + \frac{v_{\mathfrak{o}}}{a_{\mathfrak{o}}} = 13.25s \tag{7.10}$$

Every bidirectional TU $\mathfrak{m}_i$ has two neighbors (see eq. 7.1) that are directly connected to form the dynamic graphs edges $e_i \in E_\tau \in G\{0, \ldots, T\}$ later. Based on the two operations (see eq. 7.2) $\{\text{move\_left}, \text{move\_right}\}$, a container c can be moved from one end to the other, defined by the route r_c, and its stored jobs $\mathfrak{j}$ represented by a transportation sequence $\mathfrak{l}$ (see eq. 7.3) at each local scheduler $\mathfrak{S}_\mathfrak{m}$. At the beginning $\tau = 0$ and at the initialization of the system structure, all information is absent such that the data structure of $\mathfrak{m}$ is as follows:

$$\text{Scheduler} : \mathfrak{S}_\mathfrak{m} = \emptyset \tag{7.11}$$

$$\text{ReachableDestinations} : \mathfrak{R}_\mathfrak{m} = \emptyset \tag{7.12}$$

$$\text{TimeSamples} : \Delta_\mathfrak{m} = \emptyset \tag{7.13}$$

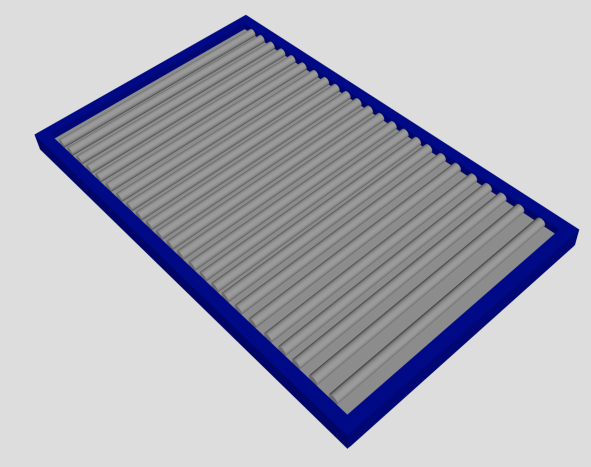


Figure 7.2: Exemplary visualization of a bidirectional transportation unit used within the system simulation.

Fig. 7.2 shows one bidirectional TU used in the simulation visualization. Physically, a roller conveyor continuously and simultaneously moves the containers c. One motor creates the needed momentum to rotated the rolls which move the containers. Each roll is connected by a fan belt either to the motor or in sequence to an already connected roll.

Position equipment

Next to bidirectional TUs, PEs (see fig. 7.1, middle) serve to position a container c to provide operations like fork and join for distribution within the system. PE's operations enable a movement in a maximum of four directions: north, east, south and west. For security reasons and to identify the exact position of a container c at each connection, a sensor block monitors the current movement. Physically, it consists of rollers for longitudinal movement (west and east directions) and roller balls for lateral movement (north and south directions). To enable a container movement in longitudinal transportation of c, roller balls have to be lowered to have direct contact with c. c is then moved in one of the two directions, in a forwards of backwards movement. In contrast, for lateral movement, the roller balls have to be raised such that the container c has direct contact with the rollers that enable this lateral movement. The movement change from forwards to backwards can be executed immediately, independent of longitudinal or lateral movement. For a corner movement, e.g. from longitudinal to lateral, an intermediate state must first lower or raise the roller balls when c is at the center of the PE.

The parametrization of one $\mathfrak{m} \in \mathfrak{M}$ as PE is the following:

$$
\begin{aligned}
\text{Neighbors} : \mathfrak{N}_\mathfrak{m} &:= \{\mathfrak{m}_j, \ldots, \mathfrak{m}_k\}, \mathfrak{m}_j \in \mathfrak{M}, \mathfrak{m}_k \in \mathfrak{M}, \\
&\quad 3 \leq |\mathfrak{N}_\mathfrak{m}| \leq 4 && (7.14) \\
\text{Operations} : \mathfrak{O}_\mathfrak{m} &:= \{\text{move\_north}, \text{move\_east}, \\
&\quad \text{move\_south}, \text{move\_west} \\
&\quad \text{lower\_rolls}, \text{raise\_rolls}\} && (7.15) \\
\text{Sequences} &: \mathfrak{l} = \mathfrak{O}_\mathfrak{m} \times \{\text{RollsUp}, \text{RollsDown}\} \times \mathfrak{O}_\mathfrak{m} \\
&\quad \{\text{RollsUp}, \text{RollsDown}\} \subseteq \mathfrak{O}_\mathfrak{m} && (7.16) \\
\text{States} : \mathfrak{S}_\mathfrak{m} &= \{WorkingNorth, WorkingEast, \\
&\quad WorkingSouth, WorkingWest, \\
&\quad RollsUp, RollsDown, Waiting\} && (7.17) \\
\text{Preconditions} : \mathfrak{O}_1, \mathfrak{O}_3 &: \text{pre}(\mathfrak{O}_1) = \text{pre}(\mathfrak{O}_2) = \mathfrak{S}_6 = RollsUp \\
\mathfrak{O}_2, \mathfrak{O}_4 &: \text{pre}(\mathfrak{O}_2) = \text{pre}(\mathfrak{O}_4) = \mathfrak{S}_7 = RollsDown \\
\mathfrak{O}_5 &: \text{pre}(\mathfrak{O}_5) = \mathfrak{S}_6 = RollsDown
\end{aligned}
$$

$$\mathfrak{O}_6 : \text{pre}(\mathfrak{O}_6) = \mathfrak{S}_5 = RollsUp \quad (7.18)$$

$$\text{Postconditions} : \mathfrak{O}_1, \mathfrak{O}_3 : \text{post}(\mathfrak{O}_1) = \text{post}(\mathfrak{O}_2) = \mathfrak{S}_6 = RollsUp$$

$$\mathfrak{O}_2, \mathfrak{O}_4 : \text{post}(\mathfrak{O}_2) = \text{post}(\mathfrak{O}_4) = \mathfrak{S}_7 = RollsDown$$

$$\mathfrak{O}_5 : \text{post}(\mathfrak{O}_6) = \mathfrak{S}_6 = RollsUp$$

$$\mathfrak{O}_6 : \text{post}(\mathfrak{O}_5) = \mathfrak{S}_5 = RollsDown \quad (7.19)$$

$$\text{Velocity} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : v_\mathfrak{o} = 0.4ms \quad (7.20)$$

$$\text{Acceleration} : \forall \mathfrak{o} \in \mathfrak{O}_\mathfrak{m} : a_\mathfrak{o} = 0.2ms^2 \quad (7.21)$$

$$\forall\{\text{move\_east}, \text{move\_west}\} \subseteq \mathfrak{O}_\mathfrak{m} : d_{\mathfrak{o},0} = 2.1m, \text{in longitudinal}$$

$$\forall\{\text{move\_south}, \text{move\_north}\} \subseteq \mathfrak{O}_\mathfrak{m} : d_{\mathfrak{o},1} = 2.8m, \text{in lateral}$$

$$\text{corner movement} : d_{\mathfrak{m},2} = 2,45m \quad (7.22)$$

$$\text{Inter. time} : \{\text{RollsUp}, \text{RollsDown}\} : \tau = 10s \quad (7.23)$$

$$\text{Static Processing Time} : \phi_{\mathfrak{m},\mathfrak{l}} = \frac{d_\mathfrak{o}}{v_\mathfrak{o}} + \frac{v_\mathfrak{o}}{a_\mathfrak{o}} = 7s, \text{for longitudinal only}$$

$$= 9s, \text{for lateral only} \quad (7.24)$$

$$\text{for corner movement} : \phi_{\mathfrak{m},\mathfrak{l}} : 18.125s \quad (7.25)$$

As shown in eq. 7.14, each PE has at least three neighbors with a maximum of four. If it has two neighbors it is classified as a bidirectional TU, and if it has one neighbor, as a source or sink. The equipment in this evaluation enables only a four-sided movement, and therefore the maximum number of neighbors is limited to four. To move a container c between two neighbors, the intermediate states $RollsDown, RollsUp$ lift or lower balls enable a 90 degree lifted movement of c. Switching costs $10s$ between the intermediate states to enable the longitudinal or lateral movement. Having more than two neighbors gives PEs the ability to fork and join paths by able to have more than 2 neighbors.

Similar to a bidirectional TU the PE starts with an empty set of the following sets:

$$\text{Scheduler} : \mathfrak{S}_\mathfrak{m} = \emptyset \quad (7.26)$$

$$\text{ReachableDestinations} : \mathfrak{R}_\mathfrak{m} = \emptyset \quad (7.27)$$

$$\text{TimeSamples} : \Delta_\mathfrak{m} = \emptyset \quad (7.28)$$

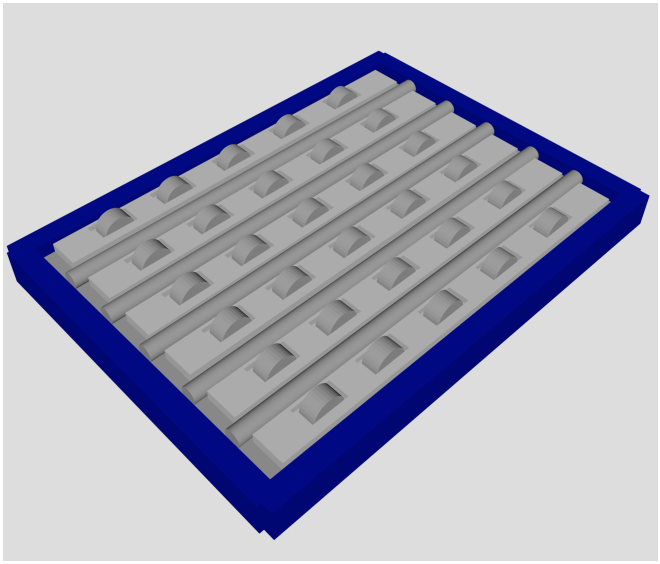


Figure 7.3: Exemplary visualization of a position equipment used within the system simulation.

Fig. 7.3 shows an exemplary PE used as a visualization during the simulation. The used rolls, similar to the bidirectional TU, are capable of longitudinal bidirectional movement. For lateral movement, the balls in the center can be raised and lowered such that they will move the container c when lifted. On field level, a total of three motors are being used to realize the movement.

Sources and sinks

Both equipment types, sources and sinks, are either bidirectional TUs or PEs with one neighbor only.

A source container arrival time can be parameterized by defining the maximum number of arriving containers c in a one-time interval. The arrival is uniformly distributed within the time interval. Either every c is randomly assigned a sink, or a fixed sink can be set.

The sink is responsible for removing a container c from the system and triggering the decentralized evaluation of the transportation of c through the system based on its path r_c and initial static reachability determination. First, each destination $\mathfrak{d}$ propagates its existence by notifying its neighbors $\mathfrak{N}_\mathfrak{d}$, which triggers a decentralized cost estimation based on the static processing time $\phi_{\mathfrak{m}_i,\mathfrak{l}}$, as explained in section 5.3.1. In summary, recursively the costs from the sink to each reachable source are recursive, cumulatively computed by adding each static processing time$\phi_{\mathfrak{m}_i,\mathfrak{l}}$ at each visited $\mathfrak{m}_i$. Every $\mathfrak{m}_i$ receiving the reachability messages from one destination $\mathfrak{d}$ from one of its neighbors $\mathfrak{m}_i$ checks

whether an entry in the reachable destination set $\mathfrak{R}_{\mathfrak{m}_i}$ already exists. If not, it is added with the cumulative static processing times from $\mathfrak{d}$ to $\mathfrak{m}_i$; otherwise, a check is done to determine which path has the shortest time, and the one with the higher costs is suppressed.

7.1.2 System composition

This section provides an overview of the used transportation system for the evaluation based on the TUs and PEs explained in the previous section 7.1.1. First, the transportation system and its parameters are introduced and based on this introduction, the specific characteristics in transportation are shown; these are deadlock and blocking caused by a container c and its route r_c. Next, the section discusses the characteristic of the system by adding and removing of TUs and PEs. The used system is an existing planned system given by an intralogistic supplier to analyze its feasibility. All data have been anonymized such that no interference with the original project can occur.

As already mentioned in the previous section, the system consists of the following four transportation types: bidirectional TUs, PEs, sources, and sinks. In short, the system model holds 909 bidirectional TUs between the sources, sinks, and PEs. In addition, 553 PEs are used to distribute the containers created by one of the 45 sources through the system to one of the 44 sinks. On average, 1,400 containers are created per hour, and randomly assigned a source and a sink.

Fig. 7.4 shows the upper part of the system. The lower part is the upper part horizontally and then vertically mirrored. Each line, line 1 to line 8, can be interpreted as a highway that allows line switching at each PE. The outer parts with their sources and sinks can only be accessed by the PEs at line 1 or line 8 in the left or right part of the system, respectively. Each outer part consists of several PEs to connect the sources and sinks with the highway in the center to reach other sinks or to be reached by other sources. Lines can only be switched between PEs such that a crossing of other containers can occur.

Fig. 7.5 displays a part of the highway with its eight lines for container distribution and the connection to one outer part. Bidirectional TUs exists between the PEs to connect them. The TUs between two PEs are folded and organized by the PEs as described in

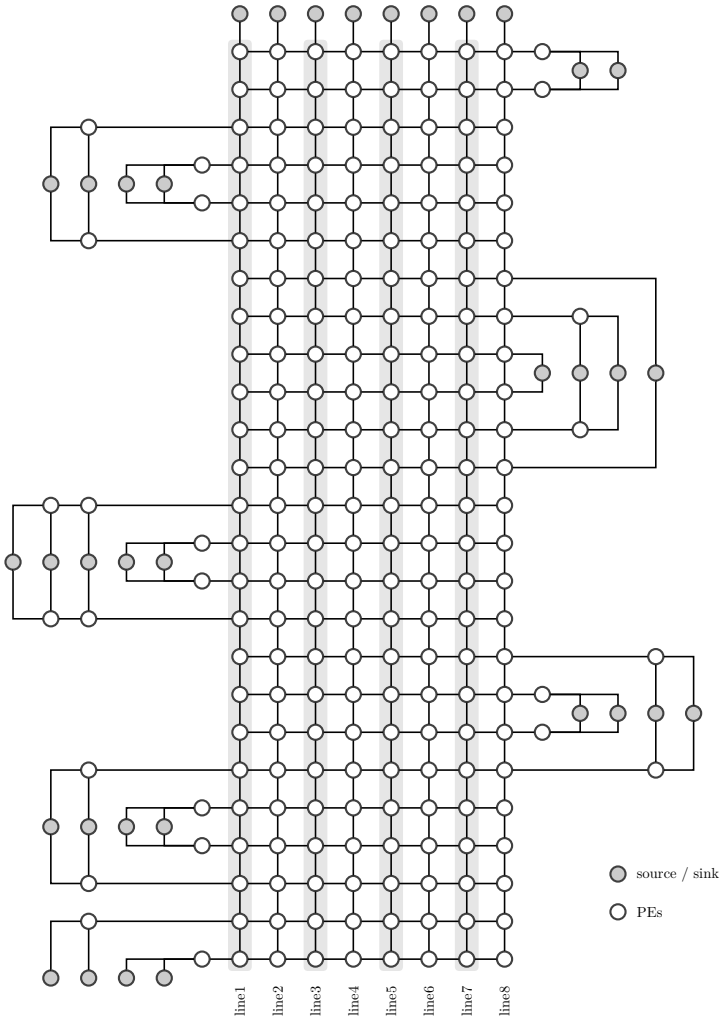


Figure 7.4: The underlying graph G_u displaying the left part of the system being evaluated.

section 5.1.1 to form the underlying graph G_u, and afterwards the first snapshot G_τ of the dynamic graph $G\{0, \ldots, T\}$ with $\tau = 0$.

As explained in section 2.1, blocking is the stumbling of one container c_1 by another container c_2 such that a detour has to be considered previously or planned during path execution of c_1. To overcome blocking, the simulation model provides several transportation lines with flow direction that can be adjusted. This degree of freedom allows the switching of directions to reduce the amount of blocking. If two inverse flows occur at one PE a distribution of container c to another line has been done, which can result in longer transportation time. A deadlock in the model is the blocking of one partial part of the system such that two or more containers cannot move through the previously computed path $r_c \in R_c$ and no other route exists in R_c. On the other hand,

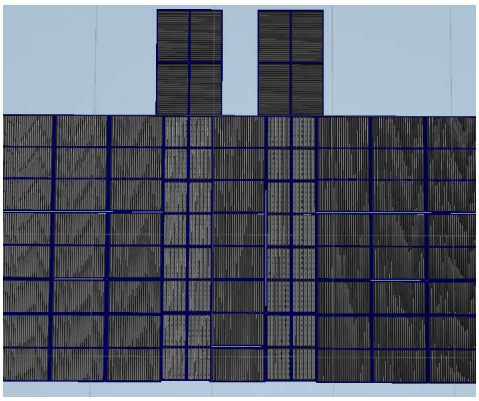


Figure 7.5: Screenshot of a part of the system model in the simulation program before start of the simulation.

the transportation of one c with r_c is called successful when c arrives at the destination $\mathfrak{d}$. During run-time, r_c is open for changes which can result in route changes.

During run-time, system changes occur by simulating the maintenance of machines. These changes are characterized by break-downs of TUs and PEs such that the lines are fully or partially inaccessible. Therefore, a break-down of one or more TUs or PEs results in the removal of possible paths and leads to the computation of a detour. Sources and sinks are always accessible by at least one route r_c. Otherwise, due to the high needed throughput, a blocking of the whole system is likely to occur because the blocked containers cannot be moved. In addition, TUs and PEs are added after an defined amount of time. These machines are then accessible for transportation again and should be considered in route computation.

7.1.3 Test environment

Each simulation has been run on the same computer in the same test environment. This environment comprises the operating system MacOS Sierra on an iMac late 2015 edition, running Unity 3D 5.6.1. The iMac has the following specification:

Processor	3,3 GHz Intel Core i5
Memory	8 GB 1867 MHz DDR3
Graphics	AMD Radeon R9 M395 2 GB

Table 7.1: Specification of the computer used for simulation.

7.2 Validation of the Protocol

The following section will focus on the validation of the protocol for decentralized opportunistic routing protocol with regard to scalability, flexibility and robustness. The scalability part demonstrates the behavior in initializing the underlying graph G_u of the system dependent on the size of sources, sinks, TUs, and PEs of the previously introduced system. G_u is used as a communication graph to compute the reachability of each PE and source / sink to ensure a correct path finding. On the other hand, the flexibility part concerns reactions based on different occurrences during the run-time. Such occurrences are the addition and removal of PEs and TUs. The method has to stabilize G_u in such a way that vertices and edges are correctly added or removed based on the occurrence. The last part focuses on robustness. Here the protocol must detect system changes and react properly to ensure a correct G_u representing the system.

7.2.1 Used parameters

The following parameters are used to validate the protocol:

- While validating the protocol, the communication layer with a ping of $20ms$ is simulated as a delay.
- It is assumed that if a sink $\mathfrak{d}$ breaks (is not functional anymore during a time window t_i, t_{i+1}), a container c will still be moved to it. An manual removal at one TU or PE where c is currently being handled is not considered.
- An equal number of sources and sinks are set during the initialization.
- No container is created during the initialization process to avoid a communication overhead while measurements are done.

7.2.2 Behavior with focus on scalability

This section focuses on the protocol‘s behavior in terms of scalability during the initial phase of creating the underlying graph G_u and its attributes as the initial costs and

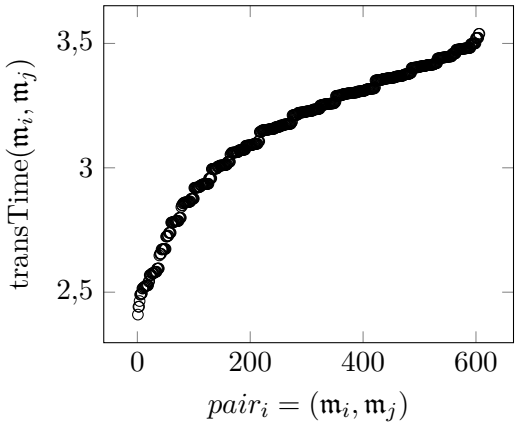


Figure 7.6: Communication time to distribute a reachable sink to a possible source.

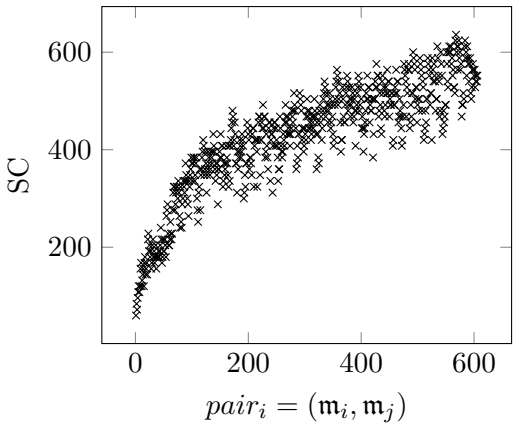


Figure 7.7: Initial static costs $\mathcal{SC}(c, \mathfrak{m}, \mathfrak{l})$ to reach a sink by a source.

reachability of sinks. As partially shown in fig. 7.4, 42 elements are used that can be switched to a source or sink respectively to τ. Each measurement ignores double pairs of a source $\mathfrak{m}_s$ and a sink $\mathfrak{m}_d$ such that if $(\mathfrak{m}_s, \mathfrak{m}_d)$ is being considered, then $(\mathfrak{m}_d, \mathfrak{m}_s)$ is ignored. Therefore, double values are not considered in the visualization of the results. The difference of a double pair is so minimal that it is considered as noise.

Fig. 7.6 shows the transmission time between a pair of a source $\mathfrak{m}_i$ and sink $\mathfrak{m}_j$ in milliseconds. During this exploration phase, the underlying graph G_u is also created by flooding the network for later efficient communication. The more closely a pair is physically connected, the faster the communication is. The left lower part represents this pairs by the sources and sinks closely connected to each other. The farther away, in terms of numbers of TUs and PEs, the sink is away from the source, the longer it takes for

exploration. Clusters, separated by the time jumps,[252] are a result of the TUs and PEs forming a highway by lines in the middle part, as shown in fig. 7.4. Each transmission between a source and sink has to go through at least one line. The clusters scale linearly within themselves. This is due to the grouping of the sources and sinks shown in the left and right parts in fig. 7.4. Breaks between the clusters show a switch between the groups. The farther away the groups are, the longer it takes for the communication and there the overall cluster communication.

$$\text{initTime} = \underset{i \in \mathfrak{M}, j \in \mathfrak{M}}{\operatorname{argmax}} \text{transTime}(i, j) \tag{7.29}$$

The maximum time for the startup is shown in eq. 7.29. To compute Init_τ ,all possible pairs of sinks and sources $(\mathfrak{m}_i, \mathfrak{m}_j)$ are evaluated by flooding the network with an initial route finding message notifying each PE of its sink existence, and computing the static costs SC from the sink $\mathfrak{m}_j$ to the source $\mathfrak{m}_i$. Each PE $\mathfrak{m}_k$ between $(\mathfrak{m}_i, \mathfrak{m}_j)$ contributes its static costs SC. If an additional message with the same sink $\mathfrak{m}_j$ arrives at $\mathfrak{m}_k$, it is dropped if the current path costs from the sink $\mathfrak{m}_j$ to the current PE $\mathfrak{m}_k$ are higher. Therefore, the initial phase ends with initTime, and afterwards, the system is operable. Based on the readability knowledge created by G_u, the initial costs for transportation can be estimated by computing the static costs only. The sum of the static transportation cost SC within a possible route r_c between a source $\mathfrak{m}_i$ and $\mathfrak{m}_j$, is used as an first indicator for route decision making during run-time. Fig. 7.6 shows the initial computed static costs between pairs $(\mathfrak{m}_i, \mathfrak{m}_j)$ of a source $\mathfrak{m}_i$ and a sink $\mathfrak{m}_j$. It is the cumulative cost of each $\mathfrak{m}_k$ between the pair $(\mathfrak{m}_i, \mathfrak{m}_j)$. During the exploration and computing of G_u each $\mathfrak{m}_k$ stores the cumulative costs from $(\mathfrak{m}_k, \mathfrak{m}_j)$ as an first indicator for the costs to reach the target $\mathfrak{m}_j$.

The pairs in fig. 7.29 are equal to those in fig. 7.6. The distributions of the static costs SC and a source and sink pair $(\mathfrak{m}_i, \mathfrak{m}_j)$ are alike. This is because the communication time depends on the number of TUs and PEs for parsing a message and for transportation. Since no other container is currently in the system and the dynamic costs are not considered, the shortest path is selected. The scattering is a result of the changing number of needed PEs, which have a significantly higher transportation time compared to TEs.

[252] A time jump is a time distance where no transmission has been done, in figure 7.6 a break without communication.

7.2.3 Behavior with focus on flexibility

The protocol for generating the underlying graph G_u and its future snapshots G_τ provides the flexibility to add or remove of vertices and edges and to change the direction of an edge. Thus, during run-time, a proper dynamic graph represents the status of the system as a snapshot G_τ. Each of the three possibilities adding, removing, and changing, is evaluated separately in the following.s

Adding

Adding means appending a vertex or edge in the system. This can occur in transportation systems by extending the current running system with additional TUs and PEs or repairing existing parts of the transportation system that is able to run again. If a TU, e.g. a regular bidirectional TU, registers its neighbors to the system via a message notifying the system of their existence, then the first PE checks whether a connection between another PE is being established, such that new paths and therefore detours can be considered. Finally, an edge is established between the two PEs $\mathfrak{m}_i$ and $\mathfrak{m}_j$ by the new TU $\mathfrak{m}_k$. Where $\mathfrak{m}_i$ and $\mathfrak{m}_j$ are two vertices $i \in V_u, j \in V_u$ in $V_u \in G_u$ and $\mathfrak{m}_k$ represents the edge $(i, j) \in E_u$.

On the other hand, if $\mathfrak{m}_k$ does not result in connecting two PEs $\mathfrak{m}_i$ and $\mathfrak{m}_j$ then no edge $(i, j) \in E_u$ is being created. Then $\mathfrak{m}_k$ is being considered to be a dead-end and therefore ignored during route finding. Theoretically speaking, $\mathfrak{m}_k$ forming a loop for a node i with $e = (i, i)$. For routing a container, this would mean in an unnecessary movement of a container back and forth such that $\mathfrak{m}_i, \mathfrak{m}_k, \mathfrak{m}_i$.

Since PEs are directly linked to vertices in G_u and later G_τ, they form the network when enabling a connection to more than two neighbors. If one connection to one PE is being linked, it is assumed that this PE is either a source, a sink, or varies between the two over time. When connecting two neighbors, a PE is being considered as a regular TU and is not transformed into one vertex $v_i \in N \in G_u$. If a PE $\mathfrak{m}_i$ is being created with a connection to at least three neighbors $(\mathfrak{m}_j, \ldots, \mathfrak{m}_n)$, it first contributes its existence in the system to its neighbors, similar to a TU, as previously explained. Afterwards, the neighbors share their connections to sinks. The PE $\mathfrak{m}_i$ contributes its local costs and distributes its new costs to the received accessible sinks. If the neighbors already have a connection to the sinks distributed by $\mathfrak{m}_i$, the message is ignored and not forwarded. Otherwise, if a new source/sink is being reachable, the information message is spread

through the network, as explained previously in the initial system process of establishing G_u. Regardless of the outcome, if a neighbor already knows the existence of the sinks spread by $\mathfrak{m}_i$, they will store the possible connection to $\mathfrak{m}_i$ for transportation. Then, after notifying, its neighbor, $\mathfrak{m}_i$ is being added to G_u and is being considered in future route finding processes.

Removing

Removing a TU or PE in the system can be either planned and or an unplanned occurrence. The planned occurrence entails informing its neighbors that a disconnect will follow. This occurs, e.g. during scheduled maintenance or shut-down due to other reasons. An unplanned occurrence is a state switch in which it must first be determined that a TU or PE cannot be reached, e.g. for transportation from an specific point in time τ which cannot be foreseen.

A planned occurrence of removing an TU or PE that relates to an edge $e_i \in E_u \in G_u$ or vertex $v_i \in V_u \in G_u$ always affects G_u and therefore future snapshots G_τ, which has a direct impact on future routes r_c. Removing, either e_i or v_i results in a reduced number of possible investigated paths for routes. This is because if $\mathfrak{m}_i \in \mathfrak{M}$ is being planned for e.g. maintenance at $\tau + n$,[253] $\mathfrak{m}_i$ will be removed from G_u at $\tau + n$. Future planned occurrences are being considered in route planning. If a route cannot be executed at $\mathfrak{m}_i$ before $\tau + n$, the route finding message is dropped. The execution time is being locally calculated at $\mathfrak{m}_i$ by using the static and dynamic costs, as explained previously. Therefore, a planned occurrence of shutting down of $\mathfrak{m}_i$ can have an impact on throughput since a potential route becomes unavailable.

On the other hand, an unplanned occurrence as a result of machine break-down, or improperly handling of machines or containers results in an immediate or prompt shut-down of the TU or PE. In contrast to a planned occurrence, an unplanned occurrence is not being considered during route finding if the occurrence at $\mathfrak{m}_i$ happened after proposing its costs to a feasible route r_c. If during route selection based on feasible routes R_c, a route r_c is selected with $\mathfrak{m}_i \notin R_c$, the route can be executed properly. Otherwise, if $\mathfrak{m}_i \in r_c$ the route cannot be executed since the container c cannot be transported over $\mathfrak{m}_i$. Periodically PEs $(\mathfrak{m}_j, \mathfrak{m}_k)$ check the existence of neighbors. Whether, a new route has to be found or there is a systematic check of its existence, the occurrence of a missing

[253] The notation $\tau + n$ is used to show that a future event occurs at the time $\tau + n$ with the current time τ.

$\mathfrak{m}_i$ will be noticed. The latest possible time is the transmission of container data before c is moved from its neighbor to $\mathfrak{m}_i$. When the missing existence of $\mathfrak{m}_i$ is being noticed, G_u is updated, and a detour is planned, as described later in sec. 7.4.4.

The removal of one TU or PE differs in whether a vertex or an edge in G_u is being removed. This has as a consequence that all future snapshots G_τ have no information about this vertex or edge. When the removal of one $\mathfrak{m}_i$ is noticed, the following steps are taken:

- Every neighbor $\mathfrak{m}_j$ removes $\mathfrak{m}_i$ as a neighbor in its neighborset $\mathfrak{N}_j$.
- If no other route to a destination $\mathfrak{d}_i$ except $\mathfrak{m}_i$ exists, the destination $\mathfrak{d}_i$ is removed at $\mathfrak{m}_j$.
- After removing a destination $\mathfrak{d}_i$, every neighbor $\mathfrak{m}_j$ notifies its neighbor that it cannot reach the destination.
- This destination removal is propagated as long as a detour exists at one $\mathfrak{m}_k$ to $\mathfrak{d}_i$.
- If a source receives the destination removal, it will no longer considers the destination as accessible.

Since every neighbor $\mathfrak{m}_j$ of $\mathfrak{m}_i$ has $\mathfrak{m}_i$ removed, the newly generated G_u no longer holds $\mathfrak{m}_i$ either.

7.2.4 Behavior with focus on Robustness

As mentioned in section 2.3 robustness of the protocol structure is defined as ensuring a stable structure by adding and removing TUs and PEs without resulting in an undefined state. A stable structure is a G_u that reflects the current system characteristics by representing its $\mathfrak{m}_i$ either as a vertex $v_i \in V_u \in G_u$ or by folded as an edge $e_i \in E_u \in G_u$. On the other hand, an undefined state is a G_u that is missing $\mathfrak{m}_i$ or where $\mathfrak{m}_j$ still exists as a vertex v_i or within an edge e_i that no longer exists, due to either planned or unplanned occurrence.

As described previously with regard to flexibility in section 7.2.3, adding or removing of $\mathfrak{m}_i$ affects the amount of $|N|, N \in G_u$ and $|E|, E \in G_u$ and an updated underlying graph G_u. Since G_u is the basis for communication, the snapshot G_τ is affected. Therefore,

new routes can be established by adding $\mathfrak{m}_i$ and routes are not considered anymore when removed.

7.3 Validation of the evaluation of transportation units

This section assesses the local behavior of each TU and PE $\mathfrak{m}_i$ by validating the self-evaluation of each $\mathfrak{m}_i$. The self-evaluation is a direct input to the static costs SC, and dynamic costs based on a future path estimation estCosts as described in section 5.2. As mentioned earlier in section 2.3, the evaluation focuses on scalability, flexibility, and robustness in terms of changes in dynamic graph $G\{0,\ldots,T\}$. Since in every snapshot $G_\tau \in G\{0,\ldots,T\}$ differs from its predecessor $G_{\tau-1}$ and successor $G_{\tau+1}$ this dynamic will be analyzed in terms of its effects on the static costs SC and dynamic costs estCosts. Then, section 7.4 draws a conclusion about global behavior from the local behavior.

First, the following explains the parameters and assumptions used for the subsequent validation. Afterwards, the behavior of the local scalability is analyzed to show that a linear computation can be ensured independently of the size of the dynamic graph. Then, the flexibility is validated to analyze the effects on local and dynamic costs of adding and removing additional $\mathfrak{m}_j$. Finally, the robustness of the costs is examined to check the feasibility of the local Scheduler $\mathfrak{S}_i$ at $\mathfrak{m}_i$.

7.3.1 Used parameters

The following parameters are used to evaluate the local behavior of TUs and PEs:

- While validating the local behavior, a ping of $20ms$ is simulated as a communication delay between two $\mathfrak{m}_i$ and $\mathfrak{m}_j$.
- If a sink $\mathfrak{d}$ breaks-down, the container c is moved on to it, to ensure that no dead[254] containers exists.
- During initialization, the number of sources and sinks is equal.
- $\phi = 1000$ as the maximum value for route length

[254]Dead containers are containers without a destination.

- $\epsilon = 1000$ as the triggering value for calculating partial routes

Due to the high value of ϕ and ϵ, no partial routes are considered. Every route r_c points directly between the source $\mathfrak{m}_i$ and the destination $\mathfrak{d}$. The focus is on the analysis of the local unit considering the effects of long routes.

7.3.2 Behavior with focus on Scalability

As mentioned in section 2.2.1, complex intralogistics systems have to handle the complexity as the number of components rises. Since every $\mathfrak{m}_i$ lacks global system information, it must to confide in a reaction to a query, e.g. route finding. Otherwise, it has to assume a break-down when another $\mathfrak{m}_j$ does not respond in time. The response is being affected by the distance between $\mathfrak{m}_i$ and $\mathfrak{m}_j$ and the local computation time of $\mathfrak{m}_j$ and every $\mathfrak{m}_k$ in between of $\mathfrak{m}_i$ and $\mathfrak{m}_j$.

Two main computations are done at $\mathfrak{m}_i$: the first one is the local static cost SC determination and the second one the dynamic cost estCost, which is an estimation of a route from $\mathfrak{m}_i$ to the destination $\mathfrak{d}$. During both computations, it must be ensured that the transportation of a container c can be executed. This transportation results in a coordination effort between direct neighbors where the container will be handed over from the predecessor $\mathfrak{m}_j$ to $\mathfrak{m}_i$, and afterwards to the successor $\mathfrak{m}_k$ of $\mathfrak{m}_i$. In addition, coordination of the underlying graph G_u can arise due to changes in the dynamic graph, as evaluated in the previous section 7.2.

First, the computation time for the static cost SC is analyzed, followed by the computation time for the dynamic cost estCost. The static costs SC are evaluated by running several routes with first increasing and secondly decreasing the amount of $\mathfrak{m}_j$. Since TUs are folded to an edge between PEs, it is shown that the complexity of the static cost SC is independent of the amount of $|\mathfrak{M}|$. Instead, it depends on the complexity of the underlying graph G_u and its number of vertices $|V_u|, V_u \in G_U$.

The static costs, as explained in section 5.2.1, are defined as $\mathrm{SC}(c, \mathfrak{m}, \mathfrak{l})$, with c as the container to be transported at $\mathfrak{m}$ with the operation sequence $\mathfrak{l}$. Assume a container c has to be moved over a PE $\mathfrak{m}_i$ to a successor $\mathfrak{m}_j$. Furthermore, $\mathfrak{l}$ connects $\mathfrak{m}_i$ and $\mathfrak{m}_j$ by a sequence of operations at $\mathfrak{m}_i$. Since $\mathfrak{l} = (\mathfrak{o}_1, \ldots, \mathfrak{o}_n)$ and every $\mathfrak{o}_i \in \mathfrak{l}$ a processing time $\phi_{\mathfrak{m}_i,\mathfrak{o}}$ is assigned, the sum $\sum_{i=1}^{n} \phi_{\mathfrak{m}_i,\mathfrak{o}}$ is the constant time c needs to be moved to $\mathfrak{m}_j$ at $\mathfrak{m}_i$. Next to the constant time the median waiting time $\widetilde{\Omega}_{\mathfrak{m}}$, is considered. Then, the time

for computing the static costs SC is dominated by the sum of operations $\mathfrak{o}_i \in \mathfrak{l}$ which is $\mathcal{O}(n)$, and the second term representing the median waiting time $\widetilde{\Omega}_{\mathfrak{m}}$ with a time of $\mathcal{O}(1)$. This results in the overall time $\mathcal{O}(1+N)$. Since a static computation time $\mathcal{O}(1)$ can be neglected, the overall time is the sum of all times ϕ for each operation $\mathfrak{o}_i \in \mathfrak{l}$, which is $\mathcal{O}(n)$ for moving one container c over $\mathfrak{m}_i$. Since the static costs SC are independent of the size of G_u and its changes G_τ, a computation can be done and stored in the initial phase for each neighbor. Then, the computation time during run-time is a search for the stored static time SC for a neighbor $\mathfrak{m}_i$ and can be interpreted as a search for the correct time. Search algorithms have been studied widely in computer science; for the sake of simplicity and easy implementation here, the linear search is used with a run-time of $\mathcal{O}(n)$ with n as the number of SC, which is equal to the number of neighbors $|N_{\mathfrak{m}_i}|$ for $\mathfrak{m}_i$. Furthermore, since each neighbor has to be either folded to an edge e_τ, e.g. bidirectional TUs connecting PEs, or a vertex v_i at snapshot G_τ, the number of neighbors at time τ is $|N_{\mathfrak{m}_i}| = |E|, E \subset E_\tau, E_\tau \in G_\tau, E = (N_i, N_j), N_i \sim \mathfrak{m}_i \vee N_j \sim \mathfrak{m}_i, N_i \in G_\tau, N_j \in G_\tau$, and therefore the complexity of searching for the static costs SC is the search task with the worst case defined by the number of edges connected to the PE $\mathfrak{m}_i$ with maximum of $\mathcal{O}(|E_\tau|)$.

After the static cost SC, the dynamic costs estCost are computed to estimate the duration of transporting c from the current investigated $\mathfrak{m}_i$ to its desired destination $\mathfrak{d}_i$. This is done in two steps: the first step is to estimate the future costs from current $v_i, v_i \in V_\tau \in G_\tau, v_i \sim \mathfrak{m}_i$ to $v_d, v_d \in V_\tau \in G_\tau, v_d \sim \mathfrak{d}$ and the second is to use linear-time slot determination to define the earliest starting time for transporting c. Therefore, both $v_i \in V_\tau \in G_\tau \wedge v_d \in V_\tau \in G_\tau$ have to be existing vertices in the current snapshot and assigned to a PE $v_i \sim \mathfrak{m}_i$ and a sink $v_d \sim \mathfrak{d}$. The first step, the estimation of the costs from the currently investigated $\mathfrak{m}_i$ to the destination $\mathfrak{d}$, is split in a lookup in the stored experienced values which will be shared after successful transportation from $\mathfrak{m}_i$ to $\mathfrak{d}$. Therefore, this lookup is done by a local search in the local routing table $\mathfrak{R}_{\mathfrak{m}_i}$ in $\mathcal{O}(log|\mathfrak{R}_{\mathfrak{m}_i}|)$. Similar to the lookup, if a destination $\mathfrak{d}$ is reached by a container c on a route r_c, the update function is done in $\mathcal{O}(log|\mathfrak{R}_{\mathfrak{m}_i}|)$. This is because the value of the destination $\mathfrak{d} \in \mathfrak{R}_{\mathfrak{m}_i}$ is updated by pastCosts(r_c) when $\mathfrak{m}_i \in r_c$. In addition, the current $\mathfrak{m}_i$ passes the message with the costs pastCost(r_c) to the neighbor $\mathfrak{m}_j$ with $\mathfrak{m}_j \in r_c$.

In summary the costs for computing the costs from $\mathfrak{m}_i$ to $\mathfrak{d}$ for the static costs SC in $\mathcal{O}(E)$ and in $\mathcal{O}(log|\mathfrak{R}_{\mathfrak{m}_i}|)$ for the dynamic costs estCost$(r_c, \mathfrak{m}_\mathrm{i})$. Since $\mathcal{O}|E| > \mathcal{O}(log|\mathfrak{R}_{\mathfrak{m}_i}|)$,

the amount needed to calculate the costs is $\mathcal{O}(|E_\tau|)$ depending on the number of edges $E_\tau \in G_\tau$.

7.3.3 Behavior with focus on flexibility

The local validation of the behavior in terms of the flexibility of one $\mathfrak{m}_i$ is split into the following:

- Reacting to a changing surrounding such that other $\mathfrak{m}_j$ are being added, $\mathfrak{m}_k$ are being removed.
- The behavior of $\mathfrak{m}_i$ is validated when entering the system by, e.g. being repaired and afterwards added to the transportation network. This case starts the execution of the initial entering protocol, as explained in section 7.2.3.
- The behavior of $\mathfrak{m}_i$ being removed from the system is not validated; it is assumed that the future local behavior is not relevant, since $\mathfrak{m}_i$ does not contribute to any future action in the system.

In addition, $\mathfrak{m}_i$, with $v_i \sim \mathfrak{m}_i$ is assumed to be a PE, a bidirectional transportation unit TE is folded to an edge e_i and being controlled locally by the connecting PEs as vertices v_i and v_j as $e_i = (v_i, v_j)$. Therefore, their behavior is limited in establishing a proper G_u resulting in a functional snapshot G_τ, as described in section 7.2.3.

The first validation is the analysis of the behavior of $\mathfrak{m}_i$ and its counterpart $v_i \sim \mathfrak{m}_i$ in a changing environment. Therefore, it is assumed that two sequential snapshots G_τ and $G_{\tau+1}$ represent a system change that directly affects $v_i \in V_\tau$ which leads to $V_\tau \nsubseteq V_{\tau+1}$ with at least one $v_i \in V_\tau \wedge v_i \notin V_{\tau+1}$ or a connected edge $e_i = (v_i, v_j), e_i \in E_\tau \nsubseteq E_{\tau+1}$. Adding one $\mathfrak{m}_j$ wherever as an edge $e_j \in E_{\tau+1}$ or a vertex $v_j \in V_{\tau+1}$ results in a change done by the protocol as described in section 7.2.3, and results in an updated G_u by establishing a communication between $v_i \sim \mathfrak{m}_i$ and $v_j \sim \mathfrak{m}_j$. On the other hand, if the newly added $\mathfrak{m}_j$ is a bidirectional TU that cannot connect an edge (v_i, v_k) with another v_j, then $\mathfrak{m}_j$ is ignored since this TU results in a dead-end for transportation. Since G_u is established and a connection exists to $v_j \sim \mathfrak{m}_j$ by an edge $e_i = (v_i, v_j)$, $\mathfrak{m}_j$ is considered along with its dynamic costs for reaching a destination $\mathfrak{d}$. All routes r_c for a container c are scheduled at $\mathfrak{S}_i$ and $\mathfrak{S}_j$ in a subsequent order, as explained in section 5.1.2. On the other hand, by removing an $\mathfrak{m}_k$ where an existing connection exists $e_k = (v_i, v_k)$ with

$v_k \sim \mathfrak{m}_k$, transportation is no longer possible. Furthermore, lets assume $\mathfrak{m}_i$ is the current PE where the container c is positioned at the current time τ. Then, $\mathfrak{m}_i$ will delete all its knowledge on $\mathfrak{m}_k$ and starts a reroute for c. In addition, the existing route r_c is removed from the scheduler $\mathfrak{S}_i$ and all future time-slots are representing other routing are untouched. This is because that it cannot be assumed that a container will arrive earlier if a route r_c can be executed when an $\mathfrak{m}_k$ breaks down.

7.3.4 Behavior with focus on robustness

This section focus on the computed local transportation time and its difference to the actual transportation time during run-time. To this end, the proposed system shown in section 7.1.2 is simulated based on a random distribution on arrival and destination selection. In total, 2,800 containers are simulated, which is the throughput of two hours in the simulation model. To measure the robustness of local cost estimation from the current $\mathfrak{m}_i$ to the destination $\mathfrak{d}$, one container moves at a time during evaluation such that no blocking blurs the transportation time cost(r_c) of container c when comparing the previously estimated time estCost(r_c) and its local static costs SC at each $\mathfrak{m}_i$.

To measure the robustness, a metric is needed to represent the difference between the real transportation time and the estimated transportation time. The following metric expresses the difference between the two:

$$\text{DiffCosts}(r_c) = \text{estDuration}(r_c) - \text{pastCost}(r_c) \tag{7.30}$$

With estDuration(r_c) as the previously estimated time for the route r_c for container c and pastCost(r_c) as the real transportation time. The outcome is as follows:

$$\begin{gathered}\text{estDuration}(r_c) = 0 : \text{Estimation is equal to transportation.} \\ \text{estDuration}(r_c) < 0 : \text{Transportation is higher than estimation.} \\ \text{estDuration}(r_c) > 0 : \text{Estimation is higher than transportation.}\end{gathered}$$

Furthermore, if:

$$\frac{100}{\text{pastCost}(r_c)} * \text{estDuration} < 5\% \tag{7.31}$$

the estimation is handled as correct. The 5% [255] threshold covers blurriness based on waiting times. Such times are shorter waiting times due to minor blocking or caused by unfinished state switches at $\mathfrak{m}_i$ for container c handling.

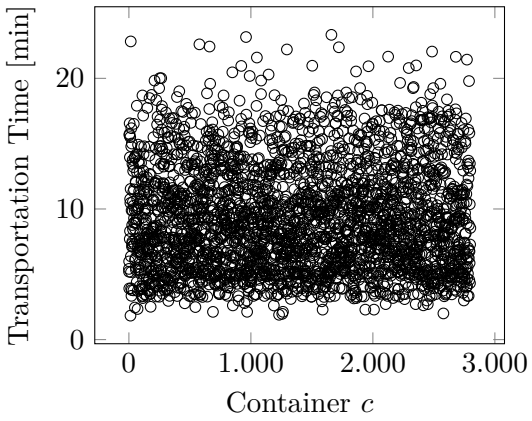


Figure 7.8: Measured transportation time for one container c during simulation.

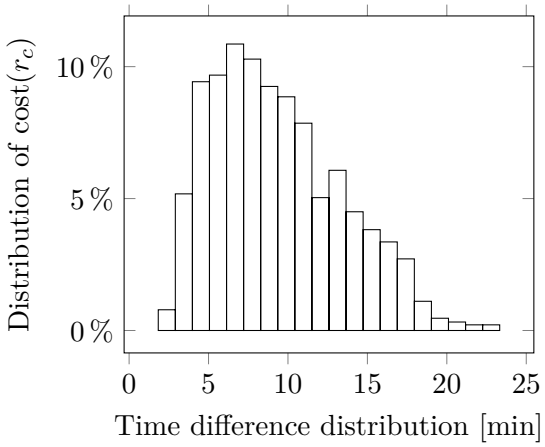


Figure 7.9: Distribution of the transportation time cost(r_c) of one container c at route r_c.

Figure 7.8 shows the transportation times of each c respectively without any collision between two container $c_i \in C$ and $c_j \in C$. Figure 7.9 shows the distribution of the transportation time. It is visually noticeable that the greater the distance $|r_c|$ between a source $\mathfrak{m}_i$ and a destination $\mathfrak{d}$ is, the fewer routes are created. The distribution suggests that the transportation time is normally distributed among all distances $|r_c|$ by all routes R. Then, the hypothesis is:

[255] 5% is chosen based on the acceptance as a significance level from [QK02], pp. 46-69 and [Fun16], pp. 1-13.

bucket:	1	2	3	4	5	6	7
range:	< 174	< 238	< 302	< 366	< 430	< 494	< 558
real:	22	139	246	293	303	280	251
expected:	128	132	144	190	233	268	288
quantities:	88.1	0.3	71.76	55.73	20.4	0.46	4.89

bucket:	8	9	10	11	12	13	14
range:	< 622	< 686	< 750	< 814	< 878	< 942	< 1006
real:	249	222	146	174	121	111	92
expected:	289	270	236	193	147	105	69
quantities:	5.6	8.8	34.8	1.94	4.76	0.34	7

bucket:	15	16	17	18	19	20
range:	< 1070	< 1134	< 1198	< 1262	< 1326	$>$
real:	81	33	14	11	5	7
expected:	43	25	13	7	3	1
quantities:	32	2.37	0.01	2.4	0.91	21.36

Table 7.2: Chi Square Test.

The transportation time is normally distributed.

This is based on the following assumption:

> Every pair of sources and destinations $(\mathfrak{m}_i, \mathfrak{d})$ has fixed transportation time similar to the sum of n, m sided dice and should be uniformly distributed.[256] Pairs and their transportation costs represent the number of dices and the cost the value one dice can reach.

Therefore, a chi square test[257] is conducted to test if the transportation time is normally distributed. As shown in fig. 7.9, 20 buckets are used to summarize the transportation time in 1 min bucket steps. To compute the respective expectancy values in seconds, the arithmetical mean of $\bar{x}_{arithm} = 560$, the standard deviation $\text{sd}[\bar{x}_{arithm}] = 244$, the minimum transportation time $\tau_{min} = 127.5$ and maximal transportation time $\tau_{max} = 1434.25$ are used.

[256] As investigated in [Gm199], [Con07], and [SR2]

[257] Pearson published the statistical hypothesis test in [Pea00] to check whether there is a significant difference between two distributions.

Then $\chi^2 = 363$ and with a degree of freedom df = 19 and significance level $p = 0.05$, the critical value is 30.144. Since $363 > 30.144$, the hypothesis that the transportation time is normally distributed can be rejected, and no further assumptions can be made comparing a normal distribution and the transportation time, such that no distribution alone can represent a transportation task.

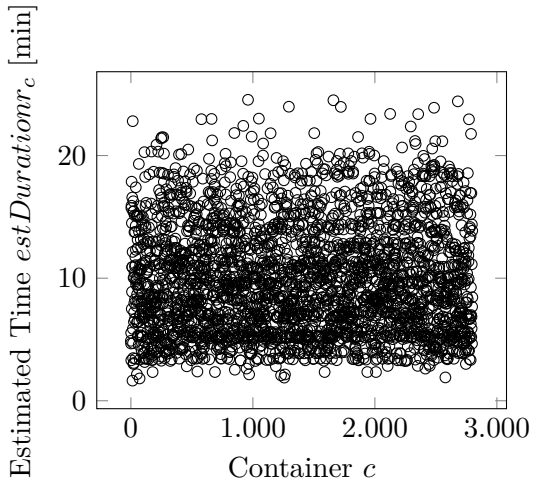


Figure 7.10: Estimated times $estDurationr_c$ for a container c for a route r_c.

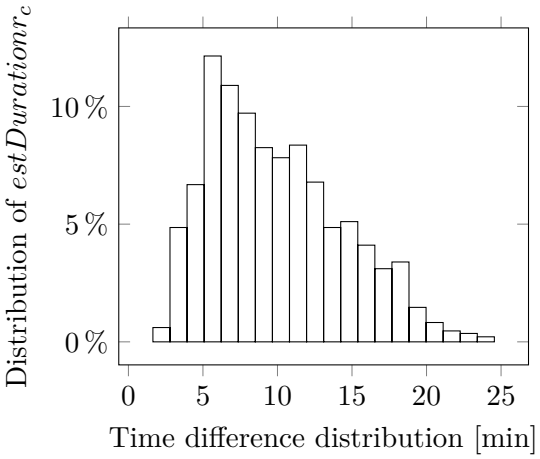


Figure 7.11: Distribution of estimated times for one container c prior routing.

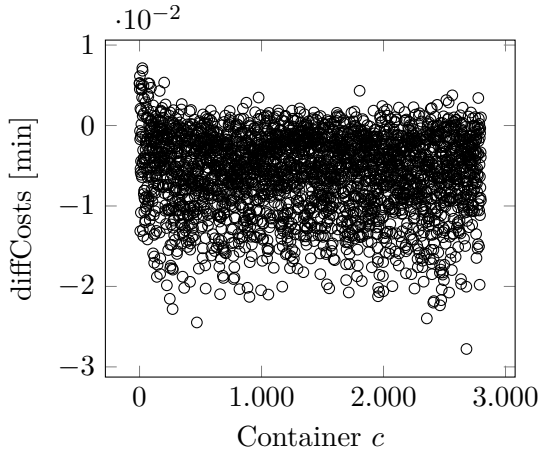


Figure 7.12: Difference between estimated time and simulation time.

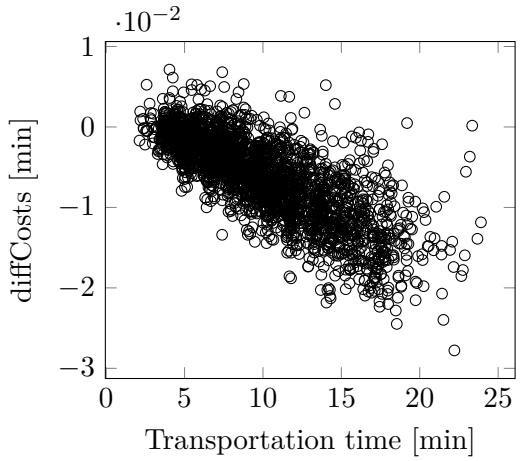


Figure 7.13: diffCosts in relation to transportation time.

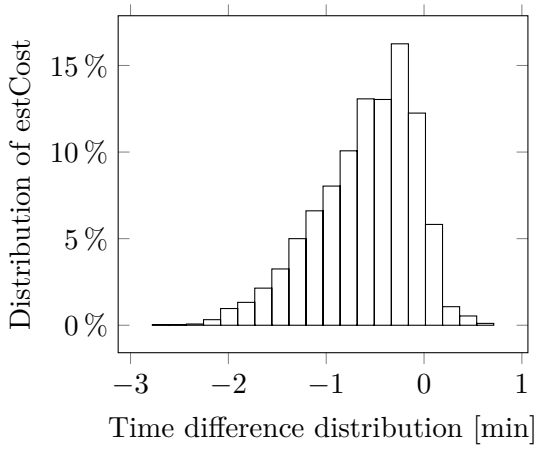


Figure 7.14: Distribution of estimated time estCost for one container c prior to routing.

Furthermore, fig. 7.10 shows the estimated time of transportation estDuration(r_c) in relation to the container c. It consists of the static costs SC representing the local movement of c at $\mathfrak{m}_i$ and its duration estCost to the destination $\mathfrak{d}$.

Fig. 7.12 shows the difference diffCosts(r_c) while fig. 7.13 shows the difference respective to the transportation time and fig. 7.14 its distribution. As previously mentioned, a correct value statement is that if the difference diffCosts(r_c) is below 5% of the transportation time, the estimated time estDuration(r_c) is being assumed to be correct. In total, 1,089 estimated transportation times are correct, representing 38.89% of all transportation tasks. It can be seen that the longer the transportation is, the greater the difference between the real transportation time pastCost(r_c) and estDuration(r_c) is. It can also be seen that the difference diffCosts(r_c) increases with the transportation time pastCost(r_c) which is directly related to the transportation duration $|r_c|$, with a minimum of -2.76 and a maximum of 0.71 minutes. This is due to the cumulative waiting time caused at $\mathfrak{m}_\mathfrak{i} \in r_c$ and state switches. Since only one container c_i at a time τ exists, it can be ruled out that any blocking by other containers c_j occurs.

Table 7.3 provides a summary of the simulation. The next part of the analysis serves to

Description	Mean	Median	Std. Deviation
route length $\|r_c\|$	47.23	44	23.99
transportation Time [min]	9.3	8.73	4.02
estCost [min]	9.9	9.33	4.42
diffCosts [min]	-0.6	-0.52	0.5
diffCosts in percentage to transportation time	-6.43 %	-5.91 %	12.39 %

Table 7.3: Summary of transportation times in the simulation to evaluate the robustness of local transportation costs.

determine the blurriness of the transportation time pastCost(r_c) which influences the estimation time estDuration(r_c).

Every transported container c at one $\mathfrak{m}_i$ has to pass several states combined in one flow $\mathfrak{f}_i$ to reach the next step $\mathfrak{m}_j$ defined in the corresponding route r_c. A flow $\mathfrak{f}_i$ consists of a sequence $\mathfrak{l}$ with states representing the low-level operations at one $\mathfrak{m}_i$ to move a container c to the next $\mathfrak{m}_j$. Therefore, the assumption is that a waiting time occurs during movement in front of $\mathfrak{m}_j$ at $\mathfrak{m}_i$ because of the current state of $\mathfrak{m}_j$ does not allow

a proper movement of c, e.g. the current state of $\mathfrak{m}_j$ is an intermediate state that is needed for preparation before handling a container c.

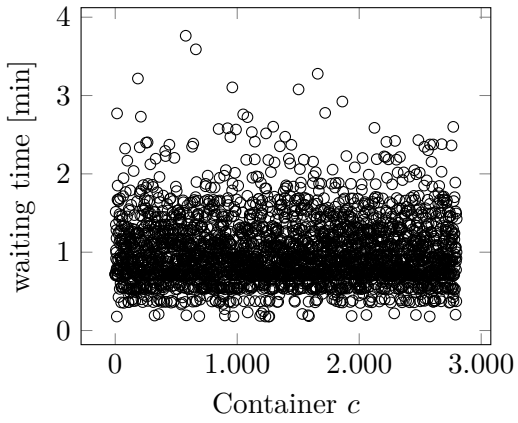


Figure 7.15: Waiting time of c during transportation with r_c.

The waiting time of c during transportation at r_c is given in fig. 7.15. Every container has to wait at least once in front of one $\mathfrak{m}_j$. The mean waiting time is 1 min and the median 0.91 min, and the standard deviation was 0.42 min. On average, the waiting time is 10.7% of the transportation time. As shown in fig. 7.16, the waiting time increases

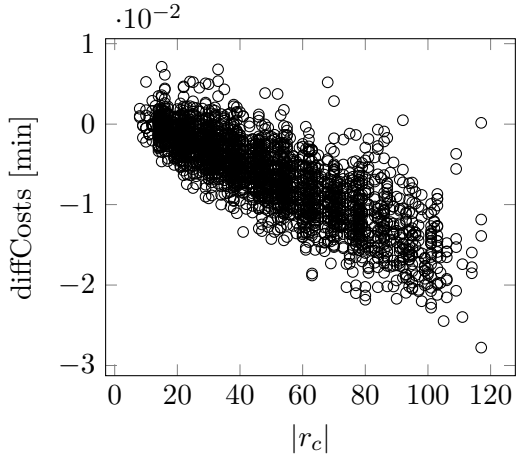


Figure 7.16: Route length $|r_c|$ in relation to the overall waiting time of c during transportation.

with the route length. This leads to the assumption that the longer the route is, the more unpredictable it becomes. The longer the route $|r_c|$ is, the larger the difference diffCosts becomes. This is due to the state switches of $\mathfrak{m}_j$ before it is able to handle c at its predecessor $\mathfrak{m}_i$. Therefore, an improvement can be established by a self-optimizing

method locally, which triggers the switching to the right handling state before c arrives.[258] In addition, fig. 7.17 shows the increasing of diffCosts in relation to the waiting time. It

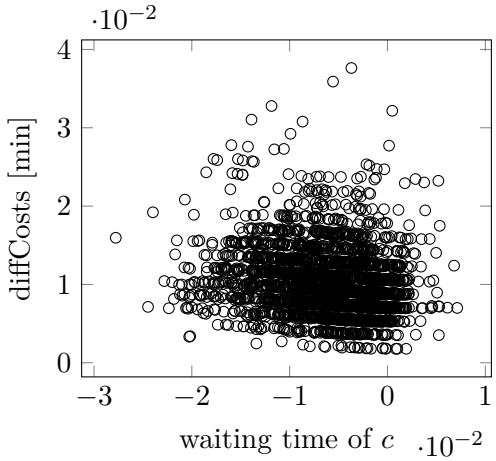


Figure 7.17: diffCosts in relation to the overall waiting time of c during transportation.

can be seen that clusters are formed and manifested as lines. Each cluster represents one group of sources and sinks in the simulation model. Both diffCosts and the waiting time grow linearly.

This section has shown that each independent transportation unit is able to evaluate itself during run-time and to contribute its knowledge to find a route. All route queries result in one route r_c which is used to transport c. However, the longer the route $|r_c|$ becomes, the more unreliable the transportation becomes. This can lead to potential blocking when inverse flows exist at one specific route. Therefore, the next section considers multiple transported containers c_i during one times-slot τ. Then, the introduced method must show that inverse flows can be handled by dissolving the situation with valid actions.

7.4 Validation of routing with local information

Having evaluated the protocol‘s functionality and the local cost estimation method, this section now assesses the routing itself. Similar to the previous sections, the focus is on scalability, flexibility, and robustness. To enable a routing, the core methods to select and execute a route r_c are first path determination, the path selection, and subsequently

[258] This low-level optimization is out of the scope of this work; which would optimize every local $\mathfrak{m}_i$ on the field level.

the execution itself. As mentioned previously, the path determination finds possible routes R_c for a container c which is evaluated among all PE $v_i \sim \mathfrak{m}_i$ such that the overall available computational power is used. Each v_i reduces the number of $|R_c|$ of possible routes in R_c for c to one candidate r_c, which represents the route $(\mathfrak{m}_i, \mathfrak{d})$ from its current position to the destination. This decision is spread to the next neighbors $\mathfrak{m}_j \in N$ and $\mathfrak{m}_j \notin r_c$, such that no recursive path exists. Each v_i to the source where the container c arrives for transportation repeats this procedure until the source makes the final decision to determine the final route $r_c \in R_c$ to use for transportation.

This section is split into three subsections. The first provides an overview of the used parameters. Afterwards, the scalability is evaluated. This is followed by a discussion of flexibility by adding and removing $\mathfrak{m}_i$ during run-time. Finally, the robustness is evaluated by analyzing the behavior of the method when transporting multiple containers C in one time-slot τ.

7.4.1 Used Parameters

To evaluate the routing, the following parameter are used:

- While validating, the local behavior a ping of $20ms$ is simulated as a communication delay between two $\mathfrak{m}_i$ and $\mathfrak{m}_j$.
- If a destination $\mathfrak{d}$ breaks down, the container c is moved to the destination, to ensure that no dead containers exists.
- During initialization, the number of sources and sinks is equal.

7.4.2 Scalability

Scalability of the routing is needed if highly complex networks are considered. An old-fashioned approach of storing and distributing all necessary information to each entity would result in either information collapse or endless storage capacity. To reduce and even improve the scalability of the necessary information for proper routing without deadlocks and with reduced blocking, two attributes have to be defined prior to system run-time. Each unit $\mathfrak{m}_i$ stores the value of both attributes and uses it to determine the

amount of spread information in the network and distance in the network relative to its position $v_i \sim \mathfrak{m}_i$.

These attributes are ϵ and ϕ as described in algorithms 3 and 6 for route determination and route reservation. Shortly, ϵ defines the maximum route length of r_c for all possible routes in the system during run-time, and ϕ sets the triggering point to start computing the next partial route at $v_i \sim \mathfrak{m}_i$ when container c is at $\mathfrak{m}_i$. The assumptions are:

- $\epsilon = \infty$, so the route r_c covers the whole length from the source $\mathfrak{m}_i$ to the destination $\mathfrak{d}$ for c.
- $\epsilon < 1$ is illegal because then only routes of length 0 would exists and no routing would be possible.
- $0 < \epsilon < \infty$ to find an optimal value that is not too high, such that a newly calculated route mostly the current status of the partial system affecting the route, and not small enough to create an high overhead in route determination.
- $\phi < 1$; after the first transportation of c at r_c, the next partial route is calculated.
- $\phi = \epsilon$, when c arrives at the end of r_c the next partial route is calculated.
- $\phi > \epsilon$, is illegal as then no additional partial route would exists and c would be stand still at the end of r_c.

These assumptions are the basis for evaluation.

To evaluate the effects of ϵ and ϕ three simulations are run. The first one is based on Dijkstra to determine the underlying costs to a destination $\mathfrak{d}$ from every $\mathfrak{m}_i$, and afterwards, A* to determine the route from a source $\mathfrak{m}_j$ to a destination $\mathfrak{d}$. Since A* is defined as $f(x) = g(x) + h(x)$ it is parameterized by $x = n_i \sim \mathfrak{m}_i$ with $g(n_i)$ as the costs from the source $\mathfrak{m}_j$ to $\mathfrak{m}_i$ and $h(\mathfrak{m}_i)$ as the costs to the destination estCosts(r_c), with r_c as the estimated costs for the route from $\mathfrak{m}_i$ to $\mathfrak{d}$. Since A* cannot react to changes over time in the graph G_τ a weighted transformation is previously defined by assigning each line in the simulation model (see fig. 7.4) a fixed direction.

The second evaluation for comparison is almost the same as the previous simulation in the previous section 7.3.4, with the difference that a fixed combination of ϵ and ϕ is evaluated. Since only one container is transported during one time-slot τ, the transportation time, estimated time, and route length are determined. Especially the route length is of interest

since it has a direct effect on the transportation time and is controllable by ϵ with the maximum route length for a partial route r_c and ϕ as the starting point at $n_i \sim \mathfrak{m}_i$ to compute the next partial route.

The last part has the same setup as the second one, with the difference that 100 containers for each fixed combination of ϵ and ϕ are been simulated. Therefore, blocking has an influence on transportation as well.

The result of all three experiments are shown in table 7.4.

Evaluation		A*	Single Container	Multi Container
Transportation Time	Mean	9.44	9.27	9.63
	Median	9.03	8.73	8.99
	Std. Deviation	4.1	4.04	4.5
Estimation	Mean	8.94	9.48	15.94
	Median	8.55	8.83	10.04
	Std. Deviation	4.25	4.17	27.37
diffCosts	Mean	0.5	-0.22	-6.31
	Median	0.54	-0.44	-0.62
	Std. Deviation	0.37	1.45	26.37
Delta in % to Trans Time	Mean	5.27	-2.33	-65.47
	Median	5.98	-5.06	-6.85
	Std. Deviation	9.02	36	585

Table 7.4: Summary of the comparison between A*, single- and multi-container simulation with best value highlighted.

In almost all cases (marked as gray), the introduced method outperforms an A* routing based on an initial Dijkstra distance computation. Mainly the standard deviation is significant lower for the delta between the estimated time and the actual transportation time, which leads to the conclusion that A* as a reference is more stable that the introduced method in terms of a prior estimation of the behavior in transportation to the introduced method.

On the other hand, the average transportation time is on average 1.8% shorter for a single container and 2.01% longer for multiple containers in average compared to A*. This higher average transportation time for multiple containers is due to the waiting estimation, which influences the costs for one route r_c to a destination $\mathfrak{d}$ during run-time. Whereas A* only takes static routes from one source $\mathfrak{m}_i$ to a destination $\mathfrak{d}$, the introduced

method varies the route r_c from similar sources to a destination based on prior waiting time at r_c. Especially the multi-container simulation shows a high standard deviation, resulting in a large difference between the average and median time estimations prior to run-time. This result is caused by high ϵ and low ϕ as a setup prior to system run-time. A high ϵ results in long routes $|r_c|$ and more unpredictability as seen in the larger difference between the transportation time and estimated time. This unpredictability is a cause of long routes that are defined once before routing. Afterwards, during routing, no changes are made to r_c as long as no $\mathfrak{m}_i$ breaks down. Thus, new containers enter during run-time and are routed along existing routes; and in a few cases, a blocking is forced to avoid deadlocks. Secondly, a low value for ϵ results in a faster computation for the additional partial paths to the destination $\mathfrak{d}$. Since ϵ is the index limiting the maximum length of r_c, the additional partial path will be computed when $\epsilon = 1$ at the first $\mathfrak{m}_i$ in r_c. Section 7.4.4 offers a deeper insight into container c blocking correlation and transportation based on existing r_c, even at fixed routes, and why a blocking resolving method is needed to ensure a deadlock-free routing. Nevertheless, this method results in additional waiting times for this kind of ϵ and ϕ constellation. Table 7.5 provides an overview of the number of partial paths in relation to ϵ and ϕ, and in addition, the transportation time can be found in table 7.6. As shown in table 7.5, with a $\epsilon \geq 45$ on average two partial routes r_{c_i} and r_{c_j} for a container c to arrive at $\mathfrak{d}$.

$\epsilon \backslash \phi$	5,00	10,00	15,00	20,00	25,00	30,00	35,00	40,00	45,00
10,00	4,82								
15,00	3,71	3,43							
20,00	2,76	2,74	2,82						
25,00	2,40	2,38	2,36	2,51					
30,00	2,18	2,28	2,32	2,20	2,23				
35,00	2,06	2,14	2,10	2,08	2,06	2,06			
40,00	2,02	2,00	2,00	2,00	2,00	2,00	2,00		
45,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	
50,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00
55,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00	2,00

Table 7.5: Average number of partial routes needed from a source $\mathfrak{m}_i$ to a destination $\mathfrak{d}$ in relation to ϵ and ϕ.

It has been shown that the scalability in route determination is determined by the combination of ϵ and ϕ which varies from project to project. There is no overall project-independent optimum for a real project, a decision must be made in terms of

$\epsilon \backslash \phi$	5	10	15	20	25	30	35	40	45
10	10.07								
15	10.30	10.01							
20	9.33	9.56	9.38						
25	9.43	8.94	8.92	10.04					
30	9.32	9.57	10.50	10.00	9.60				
35	9.82	10.02	9.69	10.06	9.56	9.35			
40	9.68	10.05	9.17	9.30	9.60	9.86	9.63		
45	9.73	9.60	9.84	9.20	9.15	10.46	9.92	9.09	
50	9.74	8.89	10.08	9.74	9.72	9.76	9.56	9.23	10.00
55	9.19	9.16	9.94	9.72	9.68	9.62	8.92	9.42	10.01

Table 7.6: Measured mean transportation time for ϕ and ϵ combination.

communication effort and transportation time, which are inversely proportional. Next, section 7.4.3 analyzes the route r_c in terms of its behavior in a dynamic system and its effects on the transportation time by adding and / or removing $\mathfrak{m}_i$ and interfering with containers during run-time.

7.4.3 Flexibility

To enable flexibility, bypasses are used when a path is exhausted, which results in a load balancing within the transportation system. During the evolving time τ, the method learns, based on the current waiting time, which partial routes not to consider until the bypass has reached a specific amount of waiting time that is higher than to the previous one.

Let two routes r_{c_1} and r_{c_2} for $c_1 \in C, c_2 \in C$ exist. As shown in the previous section, if ϵ[259] has a high value such that fewer partial routes are needed, then the probability increases that an overlapping $r_{c_1} \subset r_{c_2}$ exists, with an additional likelihood, that inverse flows occur. During routing, three scenarios cause a waiting time at $\mathfrak{m}_i \sim v_i$:

- $\mathfrak{m}_i$ is not ready for container c_1 handling; it has to switch to a certain state first.
- $\mathfrak{m}_i$ has to handle a previously arrived container c_1.
- $\mathfrak{m}_i$ operates a container c_1 moving in the opposite direction, in an inverse flow e^{-1} to e.

[259] Since ϵ defines the maximum length of all existing and future routes $|r_c|$

In the first two cases, the container c_2 waits in front of $\mathfrak{m}_i$ and can be operated in a queued manner.[260] It is assumed that the waiting time is short compared to the third case. In the latter, where an inverse flow e^{-1} exists such that $e^{-1} \in r_{c_1} \wedge e \in r_{c_2}$, the partial route between the current position of c_2 at $\mathfrak{m}_j$ and $\mathfrak{m}_i$ is being blocked for c_1 and its flow direction such that c_2 cannot move. Therefore, it is assumed that this waiting time is the longest since all operational times at each $\mathfrak{m}_k$ between $\mathfrak{m}_i$ and $\mathfrak{m}_j$ are cumulative as waiting time for c_2 before it can be transported.

Based on the insight obtained in the previous section, the constellation of ϵ and ϕ with $\epsilon = 50$ and $\phi = 10$ is set for the next simulation run. The results are shown in table 7.7. Similar to the previous multi-container simulation, a high standard deviation is measured

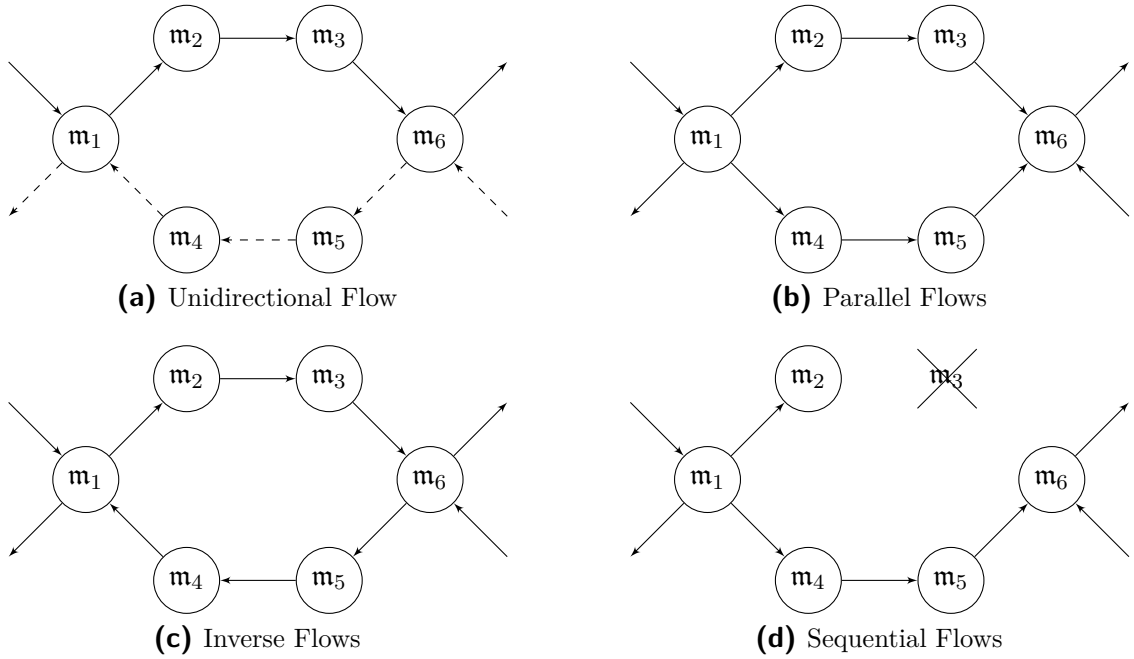
	Mean	Median	Std. Deviation
Transportation Time	9.53	8.81	4.37
diffCosts	-3.64	-0.81	8.55
Waiting Time	1.21	0.94	1.3

Table 7.7: Simulation results for flexibility analyses.

in transportation time and diffCosts. This is a result of the flexibility of the introduced method to dissolve inverse flows e^{-1} by determining the route while considering time-slots of other containers c during route determination. The consideration of every time-slot is a local optimization by determining the local static costs SC as described in section 5.2.1. First, the costs of the flow e as inbound and outbound at $\mathfrak{m}_i$ are calculated and the local scheduler S analyzed by checking any inverse e^{-1} of e. If e^{-1} exists, a check is made to see whether e can be scheduled first by checking buffer($\mathfrak{j}_{c_1,m_i}, \mathfrak{j}_{c_2,m_i}$) of the jobs $\mathfrak{j}_{c_1,m_i}$ for e and $\mathfrak{j}_{c_2,m_i}$ for e^{-1}. If so, the job is scheduled before if a free time-slot exists, and otherwise it is scheduled afterwards. In addition, the load balancing is being controlled by the waiting time W_i at $\mathfrak{m}_i$ and route time estimation estCost to the destination $\mathfrak{d}$ based on previous routing‘s pastCosts. The simulation shows that four types of load balancing occurred based on W_i and estCost.

In fig. 7.18a, only the partial example route $r_{c_1} = (\mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3, \mathfrak{m}_6)$ is being used only. This happens because estCost(r_{c_1}) is the smallest for c_1 and all future containers c_n that need to be transported between $\mathfrak{m}_1$ and $\mathfrak{m}_6$. If route r_{c_1} is overloaded due to an arbitrary number of containers causing an increase in a higher SC at $\mathfrak{m}_2$ or $\mathfrak{m}_3$, the parallel route

[260] e.g. waiting list of containers at $\mathfrak{m}_i$.



(a) Unidirectional Flow **(b)** Parallel Flows

(c) Inverse Flows **(d)** Sequential Flows

Figure 7.18: Possible flow arrangements during transportation.

$r_{c_2} = (\mathfrak{m}_1, \mathfrak{m}_4, \mathfrak{m}_5, \mathfrak{m}_6)$ is used when $\text{estCost}(r_{c_2}) < \text{estCost}(r_{c_1})$ while time τ increases the waiting time at each partial route r_{c_1} and r_{c_2} changed until it harmonizes such that during each time switch $\tau + 1$, the routing switches between r_{c_1} and r_{c_2}. In more complex transportation systems, inverse flows exists, as shown in fig. 7.18c. Here, the previously introduced partial routes become equal in terms of their transportation path but not transportation time. Whether route r_{c_1} or r_{c_2} is served first depends on its arrival time at $\mathfrak{m}_1$. The third route with an inverse flow from $\mathfrak{m}_6$ to $\mathfrak{m}_1$ through, e.g. $\mathfrak{m}_5$ and $\mathfrak{m}_4$ causes the partial path to be blocked for the time-slots $t_i, \dots t_{i+j}$ from the start-time $\alpha \in t_i$ and end-time $\beta \in t_{i+j}$. Afterwards, newly arrived routes r_{c_i} are able to redirect the path respective to their transportation path. The case in fig. 7.18d is handled almost identically to the case in fig. 7.18c, except that because there is only one existing path, each container c is handled in a first-come first-server manner, which causes high waiting time, since no convoys are being considered. Here, one path is not accessible due to a break-down of, e.g. $\mathfrak{m}_3$. If a handled container c exists from $\mathfrak{m}_2$ to $\mathfrak{m}_3$, it is rerouted through the existing path through $\mathfrak{m}_1, \mathfrak{m}_4, \mathfrak{m}_5$ and finally arrives at $\mathfrak{m}_6$. This rerouting causes a significant increase in transportation time.

In the simulation run, the average waiting time is 1.21 minutes with a standard deviation of 1.3 minutes. The higher waiting times are caused by inverse flows or sequential flows

respectively to their transportation path due to a break-down. The minimum waiting time is 0.01 minutes, caused by a unidirectional flow and a few state switches at $\mathfrak{m}_i$. The maximum of 20.19 minutes is caused by a break-down and cost-intensive rerouting at $\mathfrak{m}_j$.

Since the introduced method has the flexibility to react to certain situations such as inverse flows and break-downs of transportation units, a robustness check within inverse flows must be made to ensure a deadlock-free movement. Therefore, the next section analyzes the behavior in communication and conflict resolving when two containers c_1 and c_2 arrive at $\mathfrak{m}_i$ and $\mathfrak{m}_j$ and form an inverse flow situation and potential deadlock.

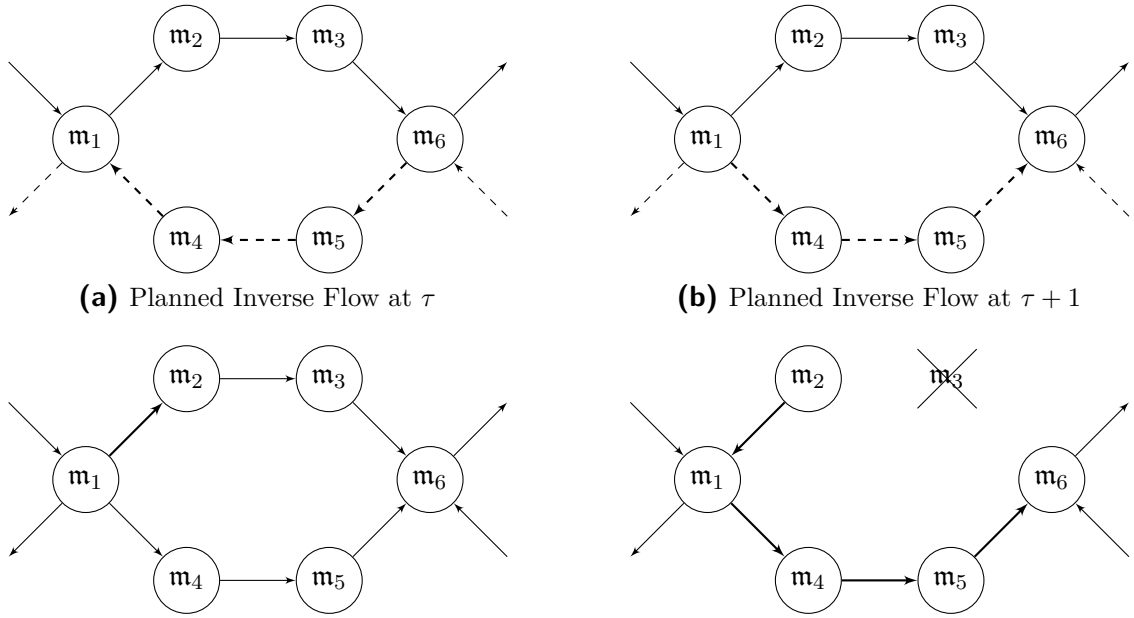
7.4.4 Robustness

Since inverse flows and break-downs exists in complex transportation systems, a part of the introduced method is responsible for transportation robustness. As introduced earlier, robustness means ensuring a deadlock-free routing while both situations, inverse flows and break-downs, occur. This is done by partial blocking of paths such that no edge can be shifted in the dynamic graph extracted from the underlying communication graph. Therefore, a part of the network is blocked for one direction only during a time slot τ.

In the simulation run, introduced in the previous section, in total 14.22% of containers were blocked due to potential deadlocks caused by regular inverse flows or break-downs. The effect was an inverse flow to dissolve the dead-end at the break-down of $\mathfrak{m}_i$. The simulation gave the insight that a distinction can be made between planned and unplanned inverse flows. Both have to be treated as the highest priority during run-time to ensure a deadlock-free system. The introduced method handles both cases as follows:

- Planned inverse flows: The containers at the partial path are moved in a first-in, first-out manner.

- Unplanned inverse flows: A rerouting occurs when $\mathfrak{m}_i$ has been investigated and a new route has been planned and new inverse flows are treated as planned inverse flows.



(a) Planned Inverse Flow at τ

(b) Planned Inverse Flow at $\tau + 1$

(c) Unplanned Inverse Flow at τ

(d) Unplanned Inverse Flow at $\tau + 1$

Figure 7.19: Cases of inverse flows handled by the introduced method.

Fig. 7.19 visualizes the two cases, planned and unplanned are visualized. First, in fig. 7.19a, the southern route $(\mathfrak{m}_6, \mathfrak{m}_5, \mathfrak{m}_4, \mathfrak{m}_1)$ is being defined in the graph G_τ within the time-slot $t = (e_i, \tau, \beta)$ such that the flow direction for the partial graph from $\mathfrak{m}_6$ to $\mathfrak{m}_1$ through $\mathfrak{m}_5$ and $\mathfrak{m}_4$ is available at least until time β. Therefore, G_β with $\tau < \beta$ is particularly unchanged. It is very likely that future containers c will be moved from $\mathfrak{m}_6$ to $\mathfrak{m}_1$ through this route. This is because the static costs SC for these four TUs are reduced since they are already in a state that allows the movement in the correct direction. Furthermore, the estimated costs to the destination $\mathfrak{d}$ from $\mathfrak{m}_6$ are smaller compared to the northern route $(\mathfrak{m}_6, \mathfrak{m}_3, \mathfrak{m}_2, \mathfrak{m}_1)$ which considered inverse flow with the consequences in a higher waiting time $W_{\mathfrak{m}_6}$. In addition, in fig. 7.19b time is advanced to $\tau + 1$ and no transportation is made from the east $\mathfrak{m}_6$ to the west $\mathfrak{m}_1$. Therefore, the south partial route edges are redirected to reduce the flow in the northern route. This is triggered by the transportation costs estCosts for routing at the northern route which is higher compared to the southern route. This happens when a specific path, e.g. northern route, is overloaded with transportation tasks. The waiting time W_i then increases at each $\mathfrak{m}_i$ and the overall costs for a route increase as well. Since the southern route is not used, a break-even point is reached such that the waiting time $W_j < W_i$ is higher.

Unplanned inverse flows behave in a similar way as planned inverse flows. The main difference is their occasion. It is not possible to foresee when they will occur. Fig. 7.19c illustrates regular transportation selection tasks. Here, during route determination, the northern route r_c is selected as the shortest route in R_c. The container c is moved until it reaches $\mathfrak{m}_2$. At this location, its successive neighbor N_i the connected TU $\mathfrak{m}_3$ breaks down such that c cannot be transported at r_c. The neighbor $\mathfrak{m}_3$ is removed from the underlying communication graph G_u and dynamic graph $G_{\tau+1}$ so that it is not being considered in any route determination, as shown in fig. 7.19d. Now, the existing container c is treated as a new arriving container $c+1$ arriving at $\mathfrak{m}_2$ at time $\tau+1$, which is the current time, or in other words, the container $c+1$ arrives immediately and has to be routed directly. The newly determined set of potential routes R_{c+1} contains at least one less route compared to the previously determined set due to the break-down of $\mathfrak{m}_3$, which formed the previously shortest route r_c. The newly determined route in r_{c+1} uses the southern route, but first the edge $e = (\mathfrak{m}_2, \mathfrak{m}_1)$ is formed in $G_{\tau+1}$ by planning the newly inverse flow. Further actions are being similar to the planned inverse flows, since due to the break-down the unplanned inverse flows are scheduled at each $\mathfrak{m}_i$ as a new route with their new times-slots for the invalid route through $\mathfrak{m}_3$ are deleted.

7.5 Evaluation conclusion

The following discusses the insight obtained from the evaluation of the introduced method regarding each sub-module in terms of scalability, flexibility, and robustness. The first part of the evaluation focused on the underlying protocol, which serves as a basis for the routing determination and execution. Afterwards, the local determination was analyzed to validate its behavior based on partial information computed without any communication to neighbors. Then, experience about local states have been estimated based on experience values, collected after successful routings. Finally, the third section focused on the final route determination by selecting the most promising candidate of feasible routes without global state knowledge of the system. The three parts are directly connected. Therefore, this section provides insight into the whole system considering the local observations in the previous sections.

Scalability

Since current transportation has become larger and more complex, a method is needed

that does not rely on centralized computational power. This monolithic concept is reaching its bottleneck in computational power and investment costs. The method introduced in this work is capable of linear scalability based on the number of used TUs for transporting a container c within the system. After the initial needed underlying graph G_u is created, the communication between $\mathfrak{m}_i$ and $\mathfrak{m}_j$ is established, and both can organize the route finding, transportation, and resolving of conflicts independently of the other part of the system. In addition, the proposed method provides the possibility of parallel computations not related to one central unit. This computation can be used for transportation and route determination separately from the rest of the system. Therefore, each unit stores its own status and information needed for these two specific tasks.

The decentralized approach leads to local optimization by only considering a partial view of the system. This optimization does not need to lead to global optimization since each $\mathfrak{m}_i$ is only interested in its local transportation execution.

Flexibility

Break-downs occur in modern transportation systems due to their high level of automation controlled by, e.g. PLCs that directly influence the motor by sensor data and high-level material-flow systems. These motors, sensors, and software are fault-prone and can cause the break-down of a whole transportation unit $\mathfrak{m}_i$. Nevertheless, the real cause of a break-down is that $\mathfrak{m}_i$ is not usable and therefore the transportation operation not executable. The introduced method handles a break-down as a new starting point for route determination and execution. All past routing information in r_c is deleted by distributing the information of a break-down and corrupt transportation path.[261] Then, the current $\mathfrak{m}_j$ where the container c is currently located triggers a new route determination, collecting all candidates R_c by acting as a new source while keeping the previous destination $\mathfrak{d}$ of c. Afterwards, in a similar way, a new route r_{c+1} is selected and being executed.

On the other hand, due to the learning mechanism based on the waiting time W_i at each $\mathfrak{m}_i$, the method reacts to overloaded paths by using parallel paths when the waiting time W_j reaches a break-even point such that $W_j < W_i$. This causes the usage of parallel paths by using the internal load balancing method.

[261] A corrupt transportation path, is a path that cannot be handled, e.g. due to a break-down.

Robustness

While scalability makes it possible, independently of size, to provide an acceptable set of routes R_c for a container to select the most promising candidate, the flexibility enables the reaction to changes during run-time by providing a detour by load balancing and rerouting if a break-down occurs. Still, especially when considering inverse flows and break-downs, each unit must react robustly when handling a solution by detouring or rerouting based on the system‘s current status. This robustness ranges from stabilizing the underlying graph G_u to extracting it to a new dynamic graph G_τ and proper execution of transportation tasks without the occurrence of deadlock. The results of this robustness is higher transportation times, as shown in section 7.4.4 in average 14% higher compared to regular transportation with a high standard deviation. Therefore, it is difficult to estimate the transportation time beforehand, without knowledge of the break-down times, which are not considered in this method.

8 Conclusion

The goal of this work was to provide a methodology to decentralize the computation and execution of dynamic transportation networks so that parts of the network‘s TUs can compute and execute routes independently based on local computation. This localized computation, which is detached from global state knowledge, enables a scalable, flexible, and robust features. The object of reflection and basis for the evaluation was a globally acting intralogistics supplier, specialized in custom-tailored transportation networks. The proposed method makes it possible to reduce the overhead in software development and to commissioning on material-flow, such that each TU is organized decentrally and operates one program only, enabling the system structure detection and operations.

8.1 Summary of the work

As introduced in sec. 2.3, a special focus was placed on the scalability, flexibility, and robustness of the decentrally operating and computing routing method. Each of the three attributes were defined and investigated for each of the three parts of the routing method. These parts are the underlying protocol as a basic structure, as defined in section 5.1; local information collecting and self-evaluation, as described in section 5.2; and finally, the determination of a set of routes, candidate selection for routing, and the execution itself, as described in section 5.3.

It was shown, that compared to the existing methods reviewed in section 3.1, the introduced method not only operates on a decentralized transportation network but is also able to operate in a decentralized manner. Thus, compared to other methods, a single unit is not responsible for information collection and decision-making which is the opposite of a centralized approach. A part of the system makes a singular decision in a decentralized manner and provides the information to affected decentralized units,

which are a part of the route determination. Later, when routing is executed, each transportation unit handles tasks independently to its responsibility such that other parts are unaffected.

8.2 Limitations

The goal of the proposed methods is to provide a decentralized computation and execution of routing in a decentrally organized dynamic transportation system. As mentioned in section 2.1, there is a heterogeneous structure of possible constellations with motors, sensors and their interactions exist. Therefore, in the introduced method the static costs SC at each $\mathfrak{m}_i$ represent the costs in a broad, abstract way such that specific peculiarities are not being reflected and causing waiting time W_i at each $\mathfrak{m}_i$. These peculiarities are the cause of waiting time when only one container is being routed at a time τ, as shown in section 7.3.4. Furthermore, the introduced method does not consider a local optimization based on future operations, such that no tasks are reorganized to increase the transportation time. The time-slot at each $\mathfrak{m}_i$ is scheduled in a first-come first-serve manner. If an available time-slot is being found it is blocked for the transportation of c without considering other transportation except inverse flows, as introduced in section 5.1.2.

Global optimization of the communication overhead is not considered. The communication and the route length which are defined with ϵ and ϕ as introduced in section 5.3.1. The results in section 7.4.3 showed that a global setting for ϵ and ϕ does not cover peculiarities of subnetworks within the dynamic network. The combination of ϵ and ϕ may be near optimal for the network as a whole for a specific time but does not have to be a local optimum in time for specific parts of the network.

The overall throughput of the system is not optimized since the scope of this work is to provide a method that enables feasible routing in a decentralized computational manner in a decentralized dynamic transportation network. In the worst-case scenario, at an edge e with an inverse flow e^{-1}, every container c is transported respectively in a zig-zag manner, such that first the container at e, then e^{-1}, e, etc. is moved. Convoys are not being considered, as shown in section 7.4.4.

8.3 Future work

Future work should examine and resolve the limits introduced in section 8.2, with a focus on optimization in terms of throughput of the method. This work has introduced the first step for a fully decentralized operating routing; in a second step, the optimization and further generalization can be built on. A further specialization would not be expedient since it would result in a specific transportation system focused on one project. Therefore, an examination of other insight of other transportation domains should provide insight into a broader generalization.

The local optimization at one TU should be an optimization of the local scheduler such that a rearrangement of tasks, e.g. to form convoys, enables higher throughput. Still, inverse flows and neighbor‘s schedulers must be considered, since parallel routes can exist. Therefore, a local optimization can be increased to partial network optimization instead.

To increase the overall throughput, a dynamic arrangement of the maximum route length and its trigger for rerouting can be localized. Each different TU could optimize its combination of length and trigger such that local peculiarities are being considered. It could be identified that this combination varies depending on the situation of the partial network being influenced by the TU. The overhead of the paths is particularly crucial.

Literaturverzeichnis

[ABH+13] Ackermann, Jörg ; Börner, Frank ; Hopf, Hendrik ; Horbach, Sebastian ; Müller, Egon: Approaches for planning and operation of adaptable factories. In: *International Journal of Production Research* 51 (2013) Nr. 15, pp. 4618–4629

[AGA99] Ascheuer, Norbert ; Grötschel, Martin ; Abdel-Hamid, Atef Abdel-Aziz: Order picking in an automatic warehouse - Solving online asymmetric TSPs.. In: *Math. Meth. of OR* 49 (1999) Nr. 3, pp. 501–515

[AIK+09] Arnold, Dieter ; Iermann, H ; Kuhn, A ; Templemeier, H ; Furmans, K: *Handbuch Logistik, 3. neubearbeitete Auflage*. Springer, 2009

[AP16] Ahmed, A M ; Paulus, R: Congestion detection technique for multipath routing and load balancing in WSN. In: *Wireless Networks* (2016)

[AR02] Allemand, M ; Royer, J C: Mixed formal specifications with PVS. In: *Proceedings 16th International Parallel and Distributed Processing Symposium*. IEEE, 2002, 8 pp

[Arn06] Arnold, Dieter: *Intralogistik*. Springer-Verlag, 2006

[BBD16] Beck, F ; Burch, M ; Diehl, S: A taxonomy and survey of dynamic graph visualization. In: *Computer Graphics ...* (2016)

[BBL12] Bloch, M ; Blumberg, S ; Laartz, J: Delivering large-scale IT projects on time, on budget, and on value. In: *Harvard Business Review* (2012)

[BCDW04] Bradbury, Jeremy S ; Cordy, James R ; Dingel, Jürgen ; Wermelinger, Michel: A survey of self-management in dynamic software architecture specifications.. In: *WOSS* (2004), pp. 28–33

[BD15] Boukerche, Azzedine ; Darehshoorzadeh, Amir: Opportunistic Routing in Wireless Networks: Models, Algorithms, and Classifications. In: *ACM Computing Surveys* 47 (2015) Nr. 2, pp. 1–36

[BHR10] Braun, T ; Heissenbüttel, M ; Roth, T: Performance of the beacon-less routing protocol in realistic scenarios. In: *Ad Hoc Networks* 8 (2010) Nr. 1, pp. 96–107

[BL95] Bonnington, C Paul ; Little, Charles H C: *The foundations of topological graph theory*. Springer-Verlag, New York, 1995

[BLM+06] BOCCALETTI, S ; LATORA, V ; MORENO, Y ; CHAVEZ, M ; HWANG, D: Complex networks: Structure and dynamics. In: *Physics Reports* 424 (2006) Nr. 4-5, pp. 175–308

[BM04] Biswas, Sanjit ; Morris, Robert: Opportunistic routing in multi-hop wireless networks.. In: *Computer Communication Review* 34 (2004) Nr. 1, pp. 69–74

[Bol09] Bolton, W: *Programmable Logic Controllers, 5th*. Elservier-Newnes, 2009

[BU12] Bradley, Donovan ; Uma, R N: Energy-efficient routing through Weighted Load Balancing.. In: *GLOBECOM* (2012), pp. 31–37

[Büs10] Büsing, Christina: *Graphen- und Netzwerkoptimierung*. Springer-Verlag, 2010

[BYJB14] Basu, Prithwish ; Yu, Feng ; Johnson, Matthew P ; Bar-Noy, Amotz: Low Expected Latency Routing in Dynamic Networks. In: *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2014, pp. 267–271

[CAG15] Chaubey, N ; Aggarwal, A ; Gandhi, S: Effect of pause time on AODV and TSDRP routing protocols under black hole attack and DoS attacks in MANETs. In: *... for Sustainable Global ...* (2015), pp. 1807–1812

[CCOR14] Carpanzano, E ; Cesta, A ; Orlandini, A ; Rasconi, R: Intelligent dynamic part routing policies in plug&produce reconfigurable transportation systems. In: *CIRP Annals- ...* 63 (2014) Nr. 1, pp. 425–428

[Cha15] Chakchouk, Nessrine: A Survey on Opportunistic Routing in Wireless Communication Networks. In: *IEEE Communications Surveys & Tutorials* 17 (2015) Nr. 4, pp. 2214–2241

[CJKK07] Chachulski, Szymon ; Jennings, Michael ; Katti, Sachin ; Katabi, Dina: Trading structure for randomness in wireless opportunistic routing.. In: *SIGCOMM* 37 (2007) Nr. 4, pp. 169–180

[Com14] Commerce, U s Department of: *Nist Special Publication 800-82 Guide to Industrial Control Systems Security*. CreateSpace, 2014

[Con07] Conroy, M: A collection of dice problems. In: (2007)

[DABM03] De Couto, Douglas S J ; Aguayo, Daniel ; Bicket, John C ; Morris, Robert: A high-throughput path metric for multi-hop wireless routing.. In: *MobiCom* (2003), p. 134

[DC12] Darehshoorzadeh, A ; Cerda-Alabern, L: *Distance progress based opportunistic routing for wireless mesh networks.(Accepted)*. Wireless communications and mobile ..., 2012

[Dec05] Decotignie, J D: Ethernet-based real-time and industrial communications. In: *Proceedings of the IEEE*. 2005

[Dij59] Dijkstra, E W: A note on two problems in connexion with graphs. In: *Numerische mathematik* 1 (1959) Nr. 1, pp. 269–271

[DIN05] DIN, E N: *61499-1, Funktionsbausteine fuer industrielle Leitsysteme—Teil 1: Architektur (IEC 61499-1: 2005); Deutsche Fassung EN 61499-1: 2005*. 2005

[DIN11] DIN: DIN EN 61784-1. In: (2011), pp. 1–271

[DSL14] Dallmeyer, J ; Schumann, R ; Lattner, A D: Don't go with the ant flow: Ant-inspired traffic routing in urban environments. In: *Journal of Intelligent ...* 19 (2014) Nr. 1, pp. 78–88

[DTOJ14] Durkop, L ; Trsek, H ; Otto, J ; Jasperneite, J: A field level architecture for reconfigurable real-time automation systems. In: *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on* (2014), pp. 1–10

[Dun05] Dunning, Gary A: *Introduction to Programmable Logic Controllers*. Cengage Learning, 2005

[DV10] Dai, Wenbin William ; Vyatkin, Valeriy: Redesign distributed IEC 61131-3 PLC system in IEC 61499 function blocks.. In: *ETFA* (2010)

[DV12] Dai, W ; Vyatkin, V: Redesign distributed PLC control systems using IEC 61499 function blocks. In: *IEEE Transactions on Automation Science ...* (2012)

[Ee11] EUROSTAT ; européenne, Union européenne Commission: *Energy, Transport and Environment Indicators*. 2011

[ElM08] ElMaraghy, Hoda: *Changeable and Reconfigurable Manufacturing Systems*. Springer Science & Business Media, 2008

[FSG10] Furmans, K ; Schoenung, F ; Gue, K R: Plug-and-work material handling systems. In: *Progress in material handling ...* (2010)

[Fun16] Fundator, Michael: Testing Ststistical Hypothesis in Light of Mathematical Aspects in Analysis of Probability. In: (2016)

[G013] Galloway, Brendan ; 0002, Gerhard P Hancke: Introduction to Industrial Control Networks.. In: *IEEE Communications Surveys & Tutorials* (2013)

[GBS13] Göpfert, I ; Braun, D ; Schulz, M: *Automobillogistik*. Springer Fachmedien Wiesbaden. 2013

[GFSU14] Gue, Kevin R ; Furmans, K ; Seibold, Zazilia ; Uludag, Onur: GridStore: A Puzzle-Based Storage System With Decentralized Control. In: *IEEE Transactions on Automation Science and Engineering* 11 (2014) Nr. 2, pp. 429–438

[GH10] Günthner, Willibald A ; Hompel, Michael: *Internet der Dinge in der Intralogistik*. Springer-Verlag, 2010

[Gm199] Gasarch, W I ; magazine, CP Kruskal Mathematics ; 1999: When Can One Load a Set of Dice so that the Sum is Uniformly Distributed?. In: *JSTOR* 72 (1999) Nr. 2, p. 133

[GR00] Günthner, W A ; Reinhart, G: *Abschlussbericht: MATVAR*. Herbert Utz Verlag, 2000

[GR14] Goebbels, Steffen ; Rethmann, Jochen: *Mathematik für Informatiker*. Springer-Verlag, 2014

[Gud13] Gudehus, Timm: *Logistik*. Springer-Verlag, 2013

[HBD15] Habib, I ; Badruddin, N ; Drieberg, M: Delay-based load-balancing routing (DLBR) algorithm for wireless ad-hoc networks. In: ... (2015), pp. 450–454

[HBQR11] Han, Mi Kyung ; Bhartia, Apurv ; Qiu, Lili ; Rozner, Eric: *O3: optimized overlay-based opportunistic routing*. ACM, 2011

[Hog02] Hogeweg, Marten: Formal specification and analysis of industrial systems. In: (2002), pp. 1–344

[HSK08] Hannelius, Tom ; Salmenperä, Mikko ; Kuikka, Seppo: Roadmap to adopting OPC UA. In: (2008), pp. 756–761

[HSS15] Hadlak, S ; Schumann, H ; Schulz, H J: A survey of multi-faceted graph visualization. In: *Eurographics Conference on ...* (2015)

[Isr09] Israel, G D: *Determining Sample Size. University of Florida IFAS extension*. 2009

[JKB03] Jovane, F ; Koren, Y ; Boër, C R: Present and future of flexible automation: towards new paradigms. In: *CIRP Annals - Manufacturing Technology* (2003)

[JMC01] Jacquet, P ; Muhlethaler, P ; Clausen, T: Optimized link state routing protocol for ad hoc networks. In: ... (2001), pp. 62–68

[JT10] John, Karl Heinz ; Tiegelkamp, Michael: *IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. 2nd. Springer Publishing Company, Incorporated, 2010

[KHJM99] Koren, Y ; Heisel, U ; Jovane, F ; Moriwaki, T: Reconfigurable manufacturing systems. In: ... *Annals-Manufacturing ...* (1999)

[KM07] Kiess, W ; Mauve, M: A survey on real-world implementations of mobile ad-hoc networks. In: *Ad Hoc Networks* (2007), pp. 324–339

[Kop11] Kopetz, Hermann: Real-Time Systems - Design Principles for Distributed Embedded Applications.. In: *Springer 2011* (2011)

[KR14] Kathiravelu, T ; Ranasinghe, N: An adaptive routing protocol with congestion avoidance for opportunistic networks. In: *Advances in ICT for Emerging ...* (2014), pp. 241–246

[KTL+06] Koenig, Sven ; Tovey, Craig A ; Lagoudakis, Michail G ; Markakis, Evangelos ; 0001, David Kempe ; Keskinocak, Pinar ; Kleywegt, Anton J ; Meyerson, Adam ; Jain, Sonal: The Power of Sequential Single-Item Auctions for Agent Coordination.. In: *AAAI* (2006)

[KWH10] Koutsonikolas, Dimitrios ; Wang, Chih-Chun ; Hu, Y Charlie: CCACK - Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments.. In: *INFOCOM* (2010)

[LGDJ12] Landsiedel, Olaf ; Ghadimi, Euhanna ; Duquennoy, Simon ; Johansson, Mikael: Low power, low delay - opportunistic routing meets duty cycling.. In: *IPSN* (2012), pp. 185–196

[LGM13] Lepuschitz, W ; Groessing, B ; Merdan, M: Evaluation of a multi-agent approach for a real transportation system. In: *... (ICIT)* (2013), pp. 1273–1278

[LLL08] Lin, Yunfeng ; Li, Baochun ; Liang, Ben: CodeOR - Opportunistic routing in wireless mesh networks with segmented network coding.. In: *ICNP* (2008)

[LLL10] Lin, Yunfeng ; Liang, Ben ; Li, Baochun: SlideOR - Online Opportunistic Network Coding in Wireless Mesh Networks.. In: *INFOCOM* (2010)

[LSS15] Lata, B T ; Sumukha, T V ; Suhas, H: SALR: Secure adaptive load-balancing routing in service oriented wireless sensor networks. In: *... and Energy Systems ...* (2015), pp. 1–5

[LZS08] Li, X ; Zhang, J ; Shi, X: CGR: Contention-Based Geographic Routing Protocol for Mobile Ad Hoc Networks. In: *2008 4th International Conference on ...* (2008), pp. 1–4

[May11] Mayer, Stephan: *Development of a Completely Decentralized Control System for Modular Continuous Conveyors*. GRIN Verlag, 2011

[MD01] Marina, M K ; Das, S R: On-demand multipath distance vector routing in ad hoc networks. In: *Network Protocols* (2001), pp. 14–23

[MK11] Mrugalska, Beata ; Kawecka-Endler, Aleksandra: Machinery Design for Construction Safety in Practice.. In: *HCI* 6767 (2011) Nr. Chapter 43, pp. 388–397

[ML85] M. G, M A ; Laughton, Say: *The Electrical Engineer's Reference Book*. 1985

[MS02] Meyers, Fred E ; Stewart, James Robert: *Motion and Time Study for Lean Manufacturing*. Pearson College Division, 2002

[NJ10] Naghshvar, Mohammad ; Javidi, Tara: Opportunistic Routing with Congestion Diversity in Wireless Multi-hop Networks.. In: *INFOCOM* (2010)

[NJE+07] Nassr, Matt S ; Jun, Jangeun ; Eidenbenz, Stephan ; Hansson, Anders A ; Mielke, Angela M: Scalable and Reliable Sensor Network Routing - Performance Study from Field Deployment.. In: *INFOCOM* (2007), pp. 670–678

[OKMS16] Othman, Akmal Aini ; Kaliani Sundram, Veera Pandiyan ; Mohamed Sayuti, Nazura ; Shamsul Bahrin, Atikah: The Relationship between Supply Chain Integration, Just-In-Time and Logistics Performance: A Supplier's Perspective on the Automotive Industry in Malaysia. In: *International Journal of Supply Chain Management* 5 (2016) Nr. 1, pp. 44–51

[OLBH12] Onori, Mauro ; Lohse, Niels ; Barata, Jose ; Hanisch, Christoph: The IDEAS project: plug & produce at shop-floor level. In: *Assembly Automation* 32 (2012) Nr. 2, pp. 124–134

[Pat98] Patzke, Robert: Fieldbus basics.. In: *Computer Standards & Interfaces* 19 (1998) Nr. 5-6, pp. 275–293

[Pea00] Pearson, Karl: *On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is Such that it Can be Reasonably Supposed to Have Arisen from Random Sampling*. 1900

[Pin12] Pinedo, Michael L: *Scheduling*. Springer Science & Business Media, 2012

[PKP13] Paulus, R ; Kumar, P D ; Philiips, P C: Performance analysis of various ad hoc routing protocols in manet using variation in pause time and mobility speed. In: *INTERNATIONAL JOURNAL OF ...* (2013)

[PLM02] Pahlavan, K ; Li, Xinrong ; Makela, J P: Indoor geolocation science and technology. In: *IEEE Communications Magazine* 40 (2002) Nr. 2, pp. 112–118

[PRD03] Pastor-Satorras, Romualdo ; Rubi, Miguel ; Diaz-Guilera, Albert: *Statistical Mechanics of Complex Networks*. Springer Science & Business Media, 2003

[QK02] Quinn, Gerry P ; Keough, Michael J: *Experimental Design and Data Analysis for Biologists*. Cambridge University Press, 2002

[RAA11] Ramzan, M ; Ali, A ; Akram, S: Formal specification of multi-agent environment using VDM-SL. In: *Computer Sciences and ...* (2011)

[RCB17] Rushton, Alan ; Croucher, Phil ; Baker, Peter: *The Handbook of Logistics and Distribution Management*. Kogan Page Publishers, 2017

[RG09] Rong, A ; Grunow, M: Shift designs for freight handling personnel at air cargo terminals. In: *Transportation Research Part E: Logistics and ...* (2009)

[RSAV16] Regulin, Daniel ; Schütz, Daniel ; Aicher, Thomas ; Vogel-Heuser, Birgit: Model based design of knowledge bases in multi agent systems for enabling automatic reconfiguration capabilities of material flow modules.. In: *CASE* (2016), pp. 133–140

[Ruo13] Ruohonen, K: *Graph theory; 2008*. 2013

[SB16] So, J ; Byun, H: Load-Balanced Opportunistic Routing for Duty-Cycled Wireless Sensor Networks. In: *IEEE Transactions on Mobile Computing* PP (2016) Nr. 99, pp. 1–1

[SCC07] Sia, Ka Cheung ; Cho, Junghoo ; Cho, Hyun-Kyu: Efficient Monitoring Algorithm for Fast News Alerts.. In: *IEEE Trans. Knowl. Data Eng.* (2007)

[SF07] Scholz-Reiter, B ; Freitag, M: Autonomous Processes in Assembly Systems. In: *CIRP Annals - Manufacturing Technology* 56 (2007) Nr. 2, pp. 712–729

[SGZJ12] Sen, Bipul ; Guo, Jun ; Zhao, Xin ; Jha, Sanjay: ECTX - A high-throughput path metric for multi-hop wireless routing exploiting MAC-layer cooperative retransmission.. In: *WOWMOM* (2012), pp. 1–9

[SKH07] Scholz-Reiter, Bernd ; Kolditz, Jan ; Hildebrandt, Torsten: Uml as a Basis to Model Autonomous Production Systems. In: *Digital Enterprise Technology*. Springer, Boston, MA, 2007, pp. 553–560

[SMA+03] Sugi, M ; Maeda, Y ; Aiyama, Y ; Harada, T ; Arai, T: A holonic architecture for easy reconfiguration of robotic assembly systems. In: *Robotics and Automation, IEEE Transactions on* 19 (2003) Nr. 3, pp. 457–464

[Spi92] Spivey, J M: *The Z notation: a reference manual. International Series in Computer Science*. Prentice-Hall, Inc. 1992

[Spu00] Spurgeon, Charles E: *Ethernet: The Definitive Guide*. "O'Reilly Media, Inc.", 2000

[SR2] Singh, A K ; Research, RJ Dalpatadu Gaming ; 2011: The Probability Distribution of the Sum of Several Dice: Slot Applications. In: *digitalscholarship.unlv.edu* ()

[Sta10] Standard, E N: *ISO 12100 (DIN EN ISO 12100) Safety of machinery CE [Electronic resource]*. 2010

[Sta11] Stanley, Richard P: *Enumerative Combinatorics:* Cambridge University Press, 2011

[Sta17] Statista: *Produktionsvolumen der Fördertechnik- und Intralogistikbranche in Deutschland in den Jahren 2010 bis 2017*. 2017

[Sto14] Stoll, T: *Dezentral gesteuerter Aufbau von Stetigförderern mittels autonomer Materialflusselemente*. 2014

[SV88] Silva, Manuel ; Valette, Robert: Petri nets and flexible manufacturing. In: *Advances in Petri Nets 1989*. Springer, Berlin, Heidelberg, 1988, pp. 374–417

[Tem05] Tempelmeier, Horst: *Material-Logistik*. Springer Science & Business, 2005

[Thr06] Thramboulidis, K: IEC 61499 in Factory Automation. In: *Advances in Computer, Information, and Systems Sciences, and Engineering: Proceedings of IETA 2005, TeNe 2005, EIAE 2005*. Springer Netherlands, 2006, pp. 115–124

[TKHP02] Toro, J ; Karppinen, J ; Hintikka, J ; Pohjanheimo, L: Automatic configuration and diagnostics for fieldbus based automation. In: *Factory Communication Systems, 2002. 4th IEEE International Workshop on* (2002), pp. 143–148

[TW13] Tanenbaum, Andrew S ; Wetherall, David J: *Computer Networks*. 2013

[Uni06] Union, E: *Richtlinie 2006/42/EG des Europäischen Parlaments und des Rates vom 17. Mai 2006 über Maschinen und zur Änderung der Richtlinie 95/16/EG (...* Amtsblatt der Europäischen Union, 2006

[Uni08] Union, E: *REGULATION (EC) No 765/2008 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 9 July 2008 setting out the requirements for accreditation and market surveillance relating to the marketing of products and repealing Regulation (EEC) No 339/93*. Amtsblatt der Europäischen Union, 2008

[UVO97] Ueda, K ; Vaario, J ; Ohkura, K: Modelling of biological manufacturing systems for dynamic reconfiguration. In: *CIRP Annals - Manufacturing Technology* (1997)

[VDI04] VDI: *Verfügbarkeit von Transport- und Lageranlagen sowie deren Teilsysteme und Elemente*. 2004

[VT93] Verwijmeren, M ; Tilanus, C B: Network planning for scheduling operations in air cargo handling: a tool in medium term goodsflow control. In: *European Journal of Operational Research* 70 (1993) Nr. 2, pp. 159–166

[Vya13] Vyatkin, Valeriy: Software Engineering in Industrial Automation: State-of-the-Art Review. In: *IEEE Transactions on Industrial Informatics* 9 (2013) Nr. 3, pp. 1234–1249

[Wal10] Wallis, W D: *A Beginner's Guide to Graph Theory*. Springer Science & Business Media, 2010

[Wat95] Waters-Fuller, N: Just-in-time purchasing and supply: a review of the literature. In: *International Journal of Operations & ...* (1995)

[Wes06] Westphal, Cédric: Opportunistic Routing in Dynamic Ad Hoc Networks - the OPRAH protocol.. In: *MASS* (2006), pp. 570–573

[WHM13] Wilfried, S ; Henrik, G ; Markus, F: Evaluation of a configuration model for the design of adaptable logistics chains in the railway vehicle manufacturing industry. In: *IFAC Proceedings Volumes* (2013)

[WIIA12] Weyns, Danny ; Iftikhar, M Usman ; Iglesia, Didac Gil de la ; Ahmad, Tanvir: A survey of formal methods in self-adaptive systems.. In: *C3S2E* (2012), pp. 67–79

[Wil96] Wilson, R J: *Introduction to Graph Theory, Vol. 4*. Peason Education Limited, 1996

[WO05] Wang, Q ; Owen, G W: Comparison between fixed-and walking-worker assembly lines. In: *Proceedings of the ...* 2005

[WS04] Wu, J ; Stojmenovic, I: Ad hoc networks. In: *COMPUTER-IEEE COMPUTER SOCIETY-* (2004)

[YNA16] Yamaguchi, K ; Nagahashi, T ; Akiyama, T: A routing based on OLSR with traffic load balancing and QoS for Wi-Fi mesh network. In: *2016 International ...* (2016), pp. 102–107

[YYWL05] Yuan, Y ; Yang, H ; Wong, SHY ; Lu, S: ROMER: resilient opportunistic mesh routing for wireless mesh networks. In: *IEEE workshop on ...* (2005)

[ZLYB07a] Zeng, K ; Lou, W ; Yang, J ; Brown III, D R: On throughput efficiency of geographic opportunistic routing in multihop wireless networks. In: *Mobile Networks and Applications* (2007)

[ZLYB07b] Zeng, K ; Lou, W ; Yang, J ; Brown III, D R: On throughput efficiency of geographic opportunistic routing in multihop wireless networks. In: *Mobile Networks and Applications* (2007)

[ZR03a] Zorzi, Michele ; Rao, Ramesh R: Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks - Multihop Performance.. In: *IEEE Trans. Mob. Comput.* (2003)

[ZR03b] Zorzi, Michele ; Rao, Ramesh R: Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks - Multihop Performance.. In: *IEEE Trans. Mob. Comput.* (2003)

[ZRBC14] Zhao, Zhongliang ; Rosário, Denis do ; Braun, Torsten ; Cerqueira, Eduardo: Context-aware opportunistic routing in mobile ad-hoc networks incorporating node mobility.. In: *WCNC* (2014)

[ZYL09] Zeng, Kai ; Yang, Zhenyu ; Lou, Wenjing: Location-Aided Opportunistic Forwarding in Multirate and Multihop Wireless Networks. In: *IEEE Transactions on Vehicular Technology* 58 (2009) Nr. 6, pp. 3032–3040