FACULTY OF ENGINEERING
AND ARCHITECTURE

Simulation framework for routing and consolidation protocols in the Physical Internet

Jules Oosterlynck
Student number: 02005324

Supervisors: Prof. dr. ir. Stijn De Vuyst, Prof. dr. ir. Birger Raa

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Industrial Engineering and Operations Research

Academic year 2021-2022

GHENT
UNIVERSITY

FACULTY OF ENGINEERING
AND ARCHITECTURE

Simulation framework for routing and consolidation protocols in the Physical Internet

Jules Oosterlynck
Student number: 02005324

Supervisors: Prof. dr. ir. Stijn De Vuyst, Prof. dr. ir. Birger Raa

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Industrial Engineering and Operations Research

Academic year 2021-2022

GHENT
UNIVERSITY

Preface

Taking a leap of faith and leaving Belgian Defence to pursue a master's degree at Ghent University was one of the scariest decisions I have ever made. Nevertheless, I can proudly say that I have thoroughly enjoyed my time in Ghent and have grown immensely both as a person and as an engineer.

I am truly humbled by the support I have received during this degree and especially while writing this master's dissertation. Therefore, I would like to extend my thanks to everyone who has contributed to its success.

Firstly, I would like to express my sincerest gratitude to my supervisors Prof. dr. ir. Stijn De Vuyst and Prof. dr. ir. Birger Raa for the superb guidance they provided during our weekly meetings. I would also like to thank them for allowing me to apply my passion for programming to one of the most important subjects in modern logistics. It has been a pleasure to geek out on design patterns and efficiency improvements.

I am also extremely thankful to my friends Anton and Réane for livening up our little office in the Plant Design room and teaching me the essentials of the Flemish culture. Without them, this process would not have been half as entertaining.

Additionally, I am immensely grateful to my parents, grandmother, sisters and brother for their support throughout my education and generally in my life.

Lastly, I would like to acknowledge my significant other, ir. Jolien Vonck, for keeping me in shape both physically and mentally, as well as for taking the time to proofread this dissertation. She really inspires me to be the best version of myself every single day.

Copyright

Explanation related to the master's dissertation and the oral presentation

"The most damaging phrase in the language is.. it's always been done this way."

Grace Hopper

Simulation framework for routing and consolidation protocols in the Physical Internet

Jules Oosterlynck

Supervisors: Prof. dr. ir. Stijn De Vuyst, Prof. dr. ir. Birger Raa

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Industrial Engineering and Operations Research

Academic year 2021-2022

Abstract

The status quo of logistics networks is ripe with opportunities for achieving efficiency improvements. The Physical Internet promises to increase the utilization of resources by uniting supply chain networks under a common set of standards. Like packet switching in the Internet, physical goods experience a multi-segment journey along standardized hubs. At each hub, autonomous protocols calculate the next destination for each item (routing). They also decide which items to pack on which vehicle and when to dispatch them (consolidation). There is currently little research on the inner workings of these protocols. This dissertation proposes an agent-based simulation framework that allows the implementation and evaluation of different protocols with varying degrees of trust and collaboration. Its code is open-sourced on GitHub. First, all steps of the development process are described in detail. This includes the modeling of network participants and a description of the implementation in Python. All assumptions and a selection of optimization decisions are thoroughly evaluated. Afterwards, the versatility of the simulation framework is demonstrated. The evaluation methodology and available diagnostic tools are presented and used to analyze and compare several protocols. This analysis starts with static routing algorithms that do not use any outside information. These approaches are then improved by allowing dynamic routing and querying information from neighboring hubs. The functionality of this framework can still be extended by relaxing assumptions. However, it is proven to be flexible and detailed enough to enable further protocol research.

Keywords: Physical Internet, protocol, simulation, routing, consolidation

Simulation framework for routing and consolidation protocols in the Physical Internet

Jules Oosterlynck

Supervisor(s): Prof. dr. ir. Stijn De Vuyst, Prof. dr. ir. Birger Raa

Abstract—**The status quo of logistics networks is ripe with opportunities for achieving efficiency improvements. The Physical Internet promises to increase the utilization of resources by uniting supply chain networks under a common set of standards. Like packet switching in the Internet, physical goods experience a multi-segment journey along standardized hubs. At each hub, autonomous protocols calculate the next destination for each item (routing). They also decide which items to pack on which vehicle and when to dispatch them (consolidation). There is currently little research on the inner workings of these protocols. This dissertation proposes an agent-based simulation framework that allows the implementation and evaluation of different protocols with varying degrees of trust and collaboration. Its code is open-sourced on GitHub. First, all steps of the development process are described in detail. This includes the modeling of network participants and a description of the implementation in Python. All assumptions and a selection of optimization decisions are thoroughly evaluated. Afterwards, the versatility of the simulation framework is demonstrated. The evaluation methodology and available diagnostic tools are presented and used to analyze and compare several protocols. This analysis starts with static routing algorithms that do not use any outside information. These approaches are then improved by allowing dynamic routing and querying information from neighboring hubs. The functionality of this framework can still be extended by relaxing assumptions. However, it is proven to be flexible and detailed enough to enable further protocol research.**

Keywords— **Physical Internet, protocol, simulation, routing, consolidation**

I. Introduction

DUE to scattered ownership and operation, an estimated 20% of trucks drives around empty and the remaining 80% only have an average load fraction of 56% [1]. The status quo of logistics networks is ripe with opportunities for achieving efficiency improvements. Furthermore, skyrocketing oil prices are putting a massive strain on global supply chains and the economy. Coupled with the bold climate targets set by nations around the world, it has never been more relevant to seek sustainable solutions.

These circumstances call for massive innovation on all fronts of logistics and transportation. Moving to electric propulsion to reduce the emissions of the vehicles themselves will not be sufficient. Industry and policymakers also needs to look for ways to do more with less resources [2]. The Physical Internet (PI) is currently the most promising initiative to improve the utilization of resources.

A. Physical Internet

The PI envisions a world where the overlapping, proprietary supply chain networks are connected by a common set of standards. A lot of its concepts are based on the Internet, which has been hugely successful in creating a network of networks. The PI aims to reduce the costs and emissions of logistics by replacing long haul transport with multi-segment journeys through multiple standardized hubs. Each companies' goods are encapsulated in standardized containers, which are then routed throughout the network like packets in the Internet. At each hub, shipments are taken over by different trucks or switched to other modes of transport. This approach promotes the consolidation of different companies' flows, which in turn should increase the average load fraction in transport vehicles.

J. Oosterlynck submitted this dissertation in order to obtain the academic degree of Master of Science in Industrial Engineering and Operations Research at Ghent University (UGent), Gent, Belgium. E-mail: jules.oosterlynck@gmail.com.

B. Protocols

Like the Internet, all decision-making should be performed by autonomous protocols. These can be responsible for different aspects of the network: (1) Containerization algorithms decide which goods to encapsulate in a container. (2) Routing protocols choose which neighbor the container should go to next. (3) Consolidation consists of deciding which containers to pack on which vehicle and when to dispatch them. [2] estimates that implementing PI should double the efficiency of current logistics systems.

II. Literature review

The first large-scale simulation project was performed by Ballot, Montreuil, Glardon, et al. in 2012 [3]. It contains three agents, each responsible for one aspect of decision-making. Their protocols do not cooperate with other agents in the network. Fully collaborative protocols have also been researched separately. [4] proposes a PI adaptation of the Internet's Border Gateway Protocol (BGP), which achieves a better performance than independent protocols. However, BGP requires a constant exchange of information between network operators. In the current ecosystem, this widespread trust between logistics providers is nowhere to be found.

ICONET [5], a research initiative that was funded by the European Commission, highlights that sharing data and operations is the most important barrier for these open logistics networks. Implementing PI will require pushing past a 'Chicken-and-egg' problem of needing shared data to collaborate and achieve efficiency improvements, but needing to prove the cost reductions to incentivize adoption.

There currently exists little research that compares the performance of PI protocols with varying degrees of trust and collaboration. This dissertation aims to help address this by providing a toolbox that allows companies to simulate how their own operations would perform in a PI setting. It can quantify the exact value of sharing specific information, which will be paramount in convincing stakeholders to invest in the PI.

III. Simulation framework

A. Modeling

Figure 1 shows the State Graph. This graph represents the PI network that is currently modeled and enables the user to easily query state information. It is automatically generated from input files. First, the NetworkX library is used to generate the topology of the network, including the weights of the edges. Each node receives two attributes: the attribute *node* contains the corresponding instance of the **Node** class, while the *trucks* attribute contains a list of instances of the **Truck** class that belong to this parent node.

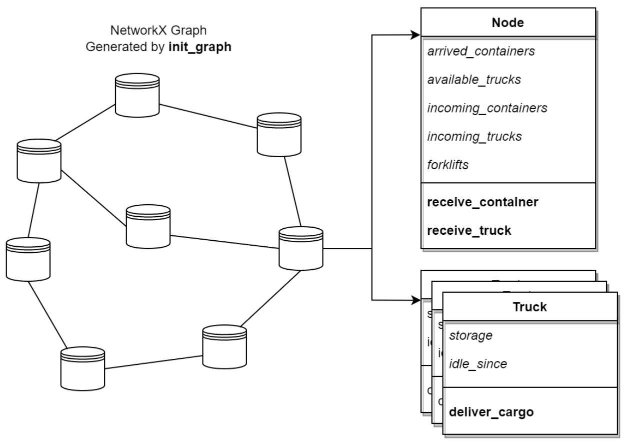


Fig. 1: Overview of simulation elements: State Graph.

The protocols can be seen as a layer built on top of this graph. Figure 2 shows the basic functioning of such a protocol. At each iteration, it queries the state information by reading some of the node's attributes. Based on these, it can decide to task a truck to deliver a cargo (set of containers) to a neighboring node. It does so by initiating the truck's **deliver_cargo** method, which then calls the destination node's methods to receive the cargo and the truck.

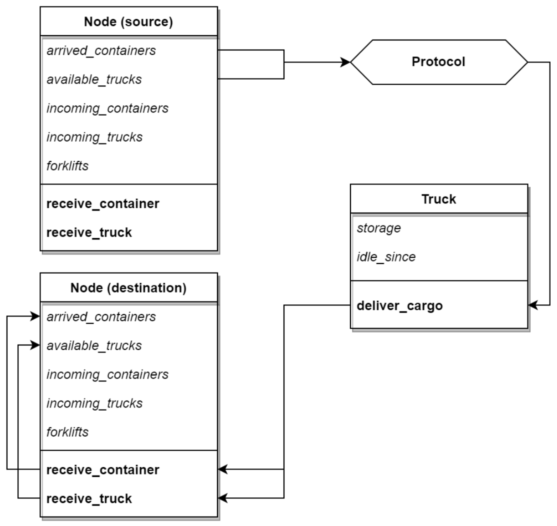


Fig. 2: Overview of simulation elements: Protocol.

B. Implementation

Since this framework aims to be customizable, it is implemented in Python. The model is simulated using the SimPy framework. This DES library is well-documented and can be used to model active agents that interact with shared resources. All documentation for this framework is embedded in the code base. Each function, class and method contains a docstring and additional comments are added to enhance readability.

The entire framework is controlled by a set of input files. These define the network topology, container flow and parameters of the simulation. All raw data is automatically saved in output files. Apart from this, the user also has the option to generate an in-depth event log.

C. Simulation

To simulate the interactions in a PI-enabled network, containers need to be generated at their starting positions. This is carried out by the Order Generator, which is depicted in figure 3. An order is defined in the input files by their start node, destination and mean interarrival time. A generator instance is assigned to each order. It simulates random arrivals following a Poisson process, defined by its mean interarrival time.

The user also has the option to populate a node's *incoming_containers* attribute with all future arrivals up to at least a *lookahead* time, which is a parameter of the simulation.

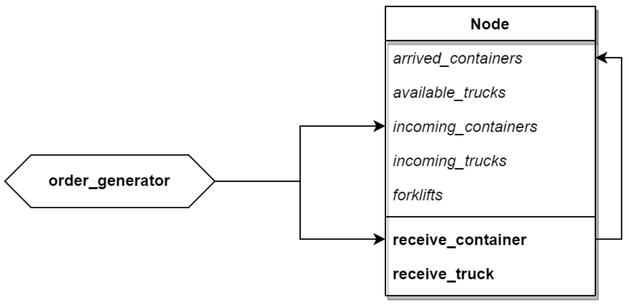


Fig. 3: Overview of simulation elements: Order Generator.

D. Assumptions

Several assumptions were made during the design of the PI network model:

- Each truck belongs to a parent node and can only drive on adjacent edges.
- The network topology is known by each node and remains constant.
- Only transport by trucks is considered.
- All containers are identical in size and are fully packed.
- Containers are an infinite resource, they are generated at their source node and destroyed upon arrival.
- Truck drivers and depot workers are available 24/7. The nodes' handling capacity never changes.
- Incoming/outgoing trucks are serviced in a FIFO queue.
- Container handling time is fixed.
- New order arrivals follow a Poisson process.

IV. Protocol design and evaluation

A. Evaluation methodology

As explained in section III-B, the simulation generates output files. These can be used to calculate several Key Performance Indicators (KPIs) in order to quantify the performance of the PI network:

- *On-time delivery fraction*: The percentage of containers that arrive at their destination before their due date.
- *Lateness*: The difference between the actual arrival time of a container and its due date. A positive lateness value means the order arrived late.
- *Total delivery time*: The total time it takes for a container to ship to its destination. This time consists of handling, transport and idle time.
- *Total truck driving time*: The sum of all truck driving times.
- *Average truck load fraction*: The average fraction of a truck's cargo that is filled with containers during transport.

Often, the KPIs differ greatly depending on the container type. To understand why these orders deviate from the rest, it is handy to visualize the position of the nodes and the flow throughout the network.

For this, a web application is written in Python using the Dash library. The output files are used to calculate flows between nodes. The visualization can be configured inside the application, as shown in figure 4. Researchers can choose which time range to consider and which flow metric to visualize.

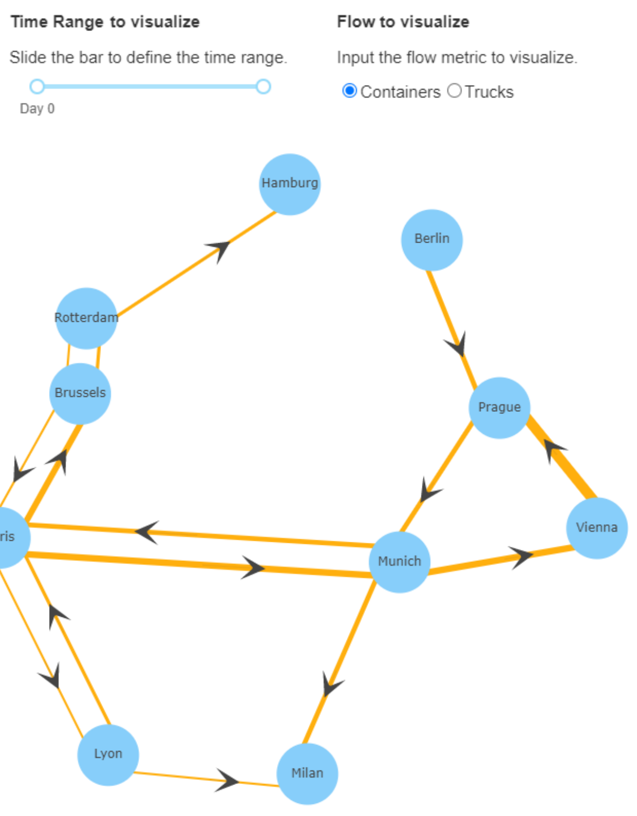


Fig. 4: Europe network, container flow visualization.

B. Generalized routing and consolidation protocol

Figure 2 showed how the protocol interacts with other elements in the simulation. Figure 5 contains a flowchart to explain its inner workings. Each protocol is first activated when the simulation starts. The first step consists of initializing several state variables that are used further on.

In the second step, the protocol updates its internal state by querying the necessary information from the State Graph. The protocol is responsible for keeping track of which containers are present at its parent node. Therefore, it is essential that the newly arrived containers are moved from the nodes' *arrived_containers* attribute into the protocol's more efficient data structure. The complexity of this data structure depends on which functionality is needed for the protocol's rules.

Once the protocol is up to date, it checks its dispatch rules. If a rule is triggered, a cargo and destination are defined and assigned to a truck using that truck's **deliver_cargo** method. These rules are sorted in order of importance: the protocol only starts checking the next rule when the current one is not triggered.

After checking all rules, the protocol uses a **yield** statement to pause its execution. The length of this pause is defined by the parameter *protocol_interval*.

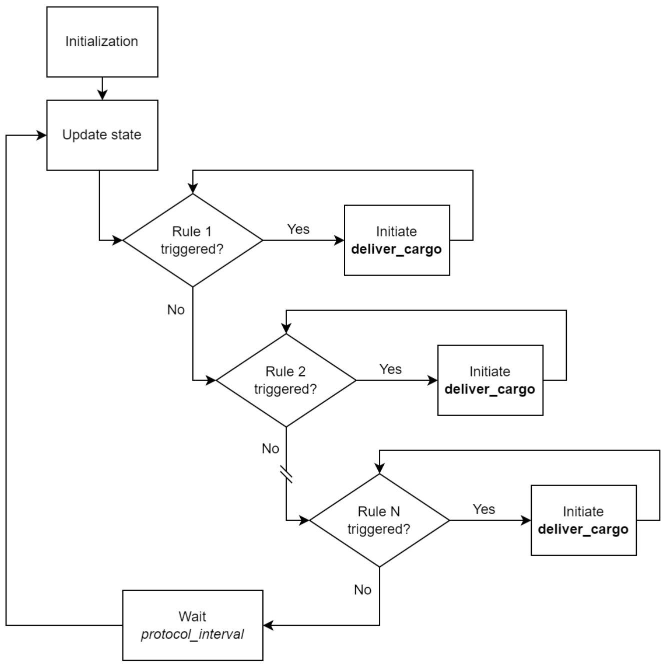


Fig. 5: Generalized routing and consolidation protocol.

C. *Static routing protocols*

Static routing protocols are first investigated in this dissertation. Assuming that the driving times between nodes is constant, these protocols do not change their routing decisions during the simulation. Four dispatch rules are developed:

1. The first rule checks if the node has any idle trucks that belong to other nodes. If this is the case, these are immediately sent back. Of course, if any container needs to go in that direction, it is first loaded onto that truck.
2. The second one dispatches a node's own trucks. It checks if there are enough containers going in the same direction to completely fill a truck's storage. If this is the case, the most urgent items are loaded and the truck is sent to that neighbor.
3. The next rule is an edit of the first rule that attempts to increase the average truck load fraction. In contrast to the first rule, it now contains a *patience* parameter. When a truck leaves its parent node and arrives at its destination, it waits up to *patience* hours before returning.
4. The last rule is urgency-based. For each container, it calculates a lower bound on the remaining transport time. It then makes sure they are sent out in time.

The first implemented protocol is volume-based. It consists of rules 1 and 2. Its performance is investigated on the Europe network depicted in figure 4. This network is simulated for a range of *orderfreq_mult* values, which is a parameter that multiplies the rate of arrivals at each order generator. This tests how the network performs with a lower/higher volume of containers. The results are depicted in figure 7 with the label 'Volume'.

This shows that the performance remains optimal between 0.5x and 1.5x the normal container volume, with an average on-time fraction of 99.27[99.03,99.51]%.

Figure 7 gives the false impression that the performance gradually decreases when the multiplier becomes larger than 1.5. This is not the case, as these higher multipliers cause the network to saturate and containers to build up. Thus, the on-time delivery fraction should actually be zero for multipliers higher than 1.5.

Even though the network should easily handle a lower load, the performance also decreases on the left side of the curve. The cause for this can be understood by inspecting figure 6.

Orders with a high mean interarrival time have a lower on-time delivery fraction. The explanation for this phenomenon is quite simple: when containers are generated at a low rate, trucks do not get dispatched quickly enough. By the time enough containers have been generated to trigger the second dispatch rule, some of them are already too late. This effect worsens when *orderfreq_mult* decreases, as this causes the arrival rates to drop.

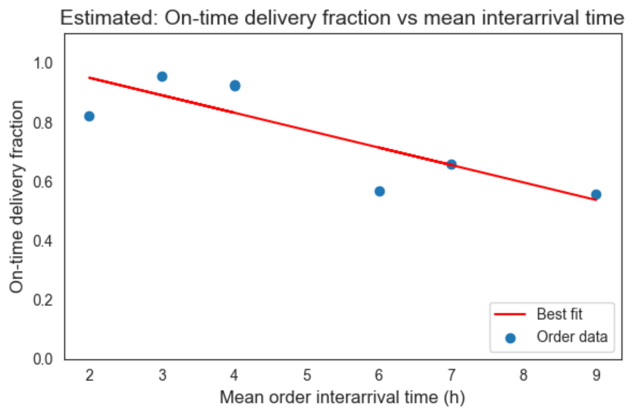


Fig. 6: Estimated: On-time delivery fraction as a function of mean interarrival time. (Network: Europe; Protocol: Volume; *orderfreq_mult*: 0.3).

The second protocol checks rule 3 and then looks at rule 2. The optimal value for the *patience* parameter is 6 hours. It increases the average truck load fraction from 56.31% to 61.87%. This in turn reduces the total driving time needed by approximately 8.4% while keeping the same on-time delivery fraction.

The third protocol contains rules 1, 2 and 4. It essentially first executes the volume-based protocol and then checks if the urgency rule is triggered. Figure 7 clearly shows the impact of adding the urgency rule to the volume protocol. The drop in on-time delivery fraction at low order frequencies has been solved. These simulations now deliver all their containers in time.

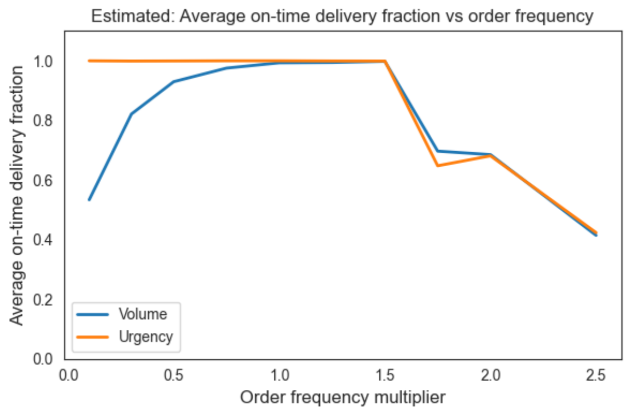


Fig. 7: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult*. (Network: Europe; Protocol: Volume and Urgency).

D. *Dynamic routing protocols*

In this last section, dynamic routing protocols are researched. Even though the assumption of constant network topology and edge weights still holds, these protocols do change their routing tables during the simulation.

The fourth protocol uses the same rules as the third, urgency-based protocol. However, it also queries the State Graph to retrieve relevant information about neighboring nodes. More specifically, it queries the *incoming_trucks* attribute of its neighbors, which contains the arrival time of every truck that is currently headed for that node. It uses this to calculate a lower bound on the time a truck would spend waiting for a forklift if it were dispatched now. This lower bound is incorporated into the driving time towards that neighbor, which can cause the routing algorithm to choose an alternative path.

This protocol is first simulated on the Europe network. It does not have any effect on its performance for two reasons: (1) The delays are often negligible compared to the distances between nodes and (2) there is a lack of alternative paths that are only slightly longer. Therefore, a new testing network is used that should benefit from this protocol. The network is appropriately named 'Test' and consists of only 4 nodes. Its topology and flows using the 'Volume' protocol are shown in figure 8.

This network is engineered specifically to trigger dynamic routing: (1) It has a large handling_time of 0.3 hours per container. (2) It contains an order from Gent to Hasselt, that can go through Brussels or Antwerp. The route through Brussels is only 0.1 hours shorter. (3) There is a large flow of containers from Hasselt to Brussels, which makes sure that Brussels has a lot of *incoming_trucks* at any time.

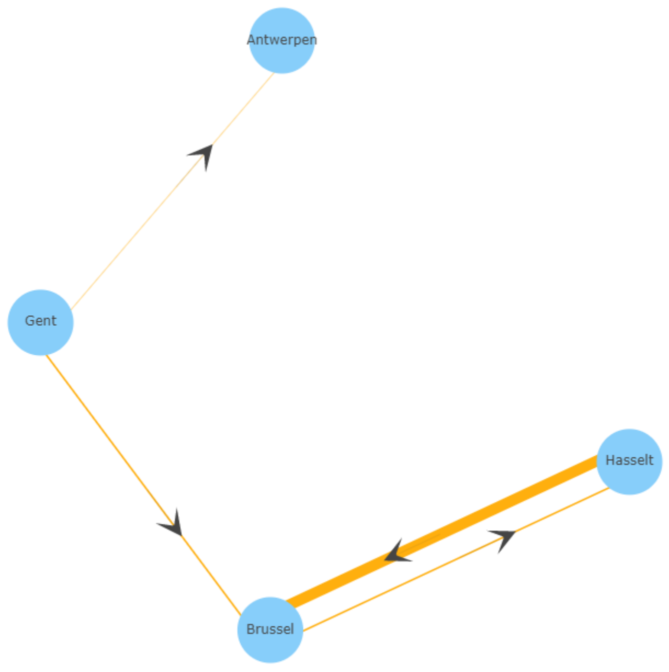


Fig. 8: Test network, container flow visualization.

The last protocol implements dynamic routing without requiring extra information. Instead, it introduces a suboptimality margin. When rule 2 is checked and no truck was able to be filled, the protocol allows containers to travel along a path that is slightly suboptimal. Setting the margin to 0.1 hours enables containers from Ghent to Hasselt to be routed through Antwerp.

The third, fourth and fifth protocol are executed for different order frequency multipliers. The results are depicted in figure 9.

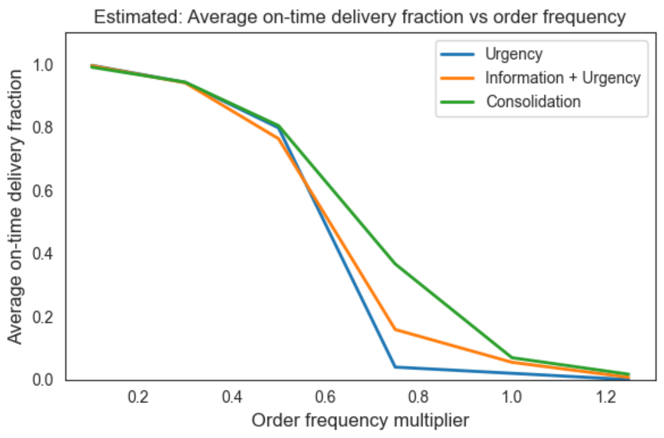


Fig. 9: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult*. (Network: Test; Protocol: Urgency, Information + Urgency and Consolidation).

Dynamic routing has no effect on the on-time delivery fraction of the network when the order frequency is low. This happens because there are never a lot of trucks being serviced at once.

Once the volume increases, the extra information significantly improves the on-time fraction. This happens because the network with the 'Urgency' protocol becomes saturated at an *orderfreq_mult* of 0.75 and higher. The new protocols successfully avoid congesting Brussels by offloading a part of the volume through Antwerp.

The fifth protocol performs the best by a wide margin, without needing access to extra information. Because there is a separate order for containers from Ghent to Antwerp, the alternative route becomes the standard. As it combines both flows, trucks towards Antwerp are often filled more quickly. This enables the alternative routing to happen at every value of *orderfreq_mult*, unlike the previous protocol. Please note that the fifth protocol only uses this alternative route because there is a separate order to Antwerp.

V. Conclusion

The primary goal of this research was to develop a modular, open-source framework for simulating PI-enabled networks. The first half of this dissertation described the development process. It started by generalizing the elements of a PI network and models these as agents. This model was then simplified by making some assumptions, which were thoroughly evaluated.

The proposed PI simulation toolbox allows research into routing and consolidation protocols. The second half of this research demonstrated this by using it to implement and analyze several protocols. First, it described the evaluation methodology and available diagnostic tools. Afterwards, a generalized rule-based protocol was defined. Next, static routing algorithms that do not use any outside information were designed and compared. The complexity was then increased by allowing dynamic routing and state information querying. From this analysis, it can be concluded that the value of certain information depends heavily on the network topology and container volume. It is thus best evaluated on a case-by-case basis.

Fortunately, this framework allows users to easily edit the network and its parameters. Even though its functionality could still be extended by relaxing assumptions, it has proven to be versatile and detailed enough to enable PI protocol research.

VI. Future research

This section provides suggestions for future PI research that can leverage the proposed simulation framework.

A. *Assumptions*

The simulation toolbox can be improved by editing or fully relaxing some of the assumptions. Doing so could lead to various future research avenues:

- The network topology should not be constant. In reality, the driving time between two places is a random variable with a long tail to account for large disturbances like accidents and traffic jams. Secondly, the shape of this distribution greatly depends on the time of day and even on the period of the year.
- The current model only supports trucks. Enabling multimodal transport is however one of the main selling points of the PI. Protocols should decide which vehicle to use based on certain criteria like cost and emissions.
- Instead of assuming all containers are identical and fully packed, a containerization protocol could be developed. This chooses the size of the container and fills it with goods. Another agent is then necessary to optimally fill transport means with these containers.
- This simulation does not contain any capabilities to manage empty containers. Nevertheless, it is an important and complex research area that has not yet been fully examined in a PI setting. Future investigations could focus on building protocols for container fleet management and analyzing the influence of the total amount of containers.
- The number of available workers could be simulated as a random variable. Furthermore, driving time constraints for drivers could be added to the consolidation protocols.
- The handling time estimate could be improved by accounting for economies of scale.

B. *PI network design*

The second category of future research topics covers the design of PI networks and its effect on the KPIs. The placements and sizes of hubs and the connections between them have a large impact on the performance of the network. These set the boundaries within which protocols can route containers to their destination. It could be interesting to research a formalized version of this problem: *Given a set of overlapping supply chains with historical container flows, what is the optimal way to combine these in a PI setting?*

Furthermore, the influence of demand unbalance on PI networks is currently not well understood. Networks where only one side is producing goods will probably not benefit as much from the PI, as most trucks will continue returning empty. It could be beneficial to formulate a parameter that quantifies how balanced the demand is. The influence of this parameter can then be studied by using the proposed simulation framework.

C. *PI protocol research*

The final recommended subject is a direct follow-up of the analysis in this dissertation. The literature review revealed that there is no existing research on the effect of using protocols with varying degrees of trust and collaboration. The framework could be used to investigate the following two topics:

- There are still a lot of approaches left to explore with rule-based protocols that only manage a single node. For example, a protocol can be developed that queries not only the list of future truck arrivals, but also the future arrivals of newly generated containers. This information can be used to obtain a more detailed estimate of the idle time at neighboring hubs. The influence of the *lookahead* value could also be examined.
- This dissertation did not yet implement any large-scale optimization protocols. These control multiple nodes at once, or even the entire network. Doing so, they could achieve a higher level of collaboration and plan departures in advance.

Acknowledgments

The author would like to acknowledge the supervisors Prof. dr. ir. Stijn De Vuyst and Prof. dr. ir. Birger Raa for the superb guidance they provided.

References

[1] T. Ambra, *The physical internet: A next-gen vision on logistics*, 2021. [Online]. Available: https://www.imec-int.com/en/articles/physical-internet-next-gen-vision-logistics.

[2] E. Ballot, S. Barbarino, B. van Bree, *et al.*, *Roadmap to the physical internet, executive version*, 2020. [Online]. Available: https://www.etp-logistics.eu/wp-content/uploads/2020/11/Roadmap-to-Physical-Intenet-Executive-Version_Final.pdf.

[3] E. Ballot, B. Montreuil, R. Glardon, *et al.*, "Simulation de l'internet physique: Contribution à la mesure des enjeux et à sa définition," *PREDIT report to the 'Ministère de l'Écologie, de l'énergie, du Développement durable et de la Mer en charge des Technologies vertes et des Négociations sur le climat' of France*, p. 206, 2012.

[4] S. Gontara, A. Boufaied, and O. Korbaa, "Routing the pi-containers in the physical internet using the pi-bgp protocol," Oct. 2018, pp. 1–8. DOI: 10.1109/AICCSA.2018.8612885.

[5] ICONET Consortium, *Living lab 4: Physical internet as enabler towards networked warehousing as a service (waas)*, 2021. [Online]. Available: http://www.iconetproject.eu/wp-content/uploads/2021/04/ICONET-Living-Lab-4-Warehousing-as-a-Service.pdf.

Contents

List of Abbreviations and Symbols

ABS	Agent-Based Simulation
BGP	Border Gateway Protocol
DES	Discrete Event Simulation
DMU	Decision Making Unit
FIFO	First In, First Out
FMCG	Fast-Moving Consumer Goods
IoT	Internet of Things
IP	Internet Protocol
KPI	Key Performance Indicator
MILP	Mixed Integer Linear Programming
OLI	Open Logistics Interconnection
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PI or π	Physical Internet
SD	System Dynamics
TCP	Transmission Control Protocol

Chapter 1

Introduction

1.1 Problem statement

Due to scattered ownership and operation, an estimated 20% of trucks drives around empty and the remaining 80% only have an average load fraction of 56% [1]. The status quo of logistics networks is ripe with opportunities for achieving efficiency improvements. Furthermore, skyrocketing oil prices are putting a massive strain on global supply chains and the economy. Coupled with the bold climate targets set by nations around the world, it has never been more relevant to seek sustainable solutions.

These circumstances call for massive innovation on all fronts of logistics and transportation. Moving to electric propulsion to reduce the emissions of the vehicles themselves will not be sufficient. Industry and policymakers also needs to look for ways to do more with less resources [2]. The Physical Internet (PI) is currently the most promising initiative to improve the utilization of resources.

1.2 Physical Internet fundamentals

The PI envisions a world where the overlapping, proprietary supply chain networks are connected by a common set of standards. A lot of its concepts are based on the Internet, which has been hugely successful in creating a network of networks. The PI aims to reduce the costs and emissions of logistics by replacing long haul transport with multi-segment journeys through multiple standardized hubs. An illustration of this evolution is provided in figures 1.1 and 1.2.

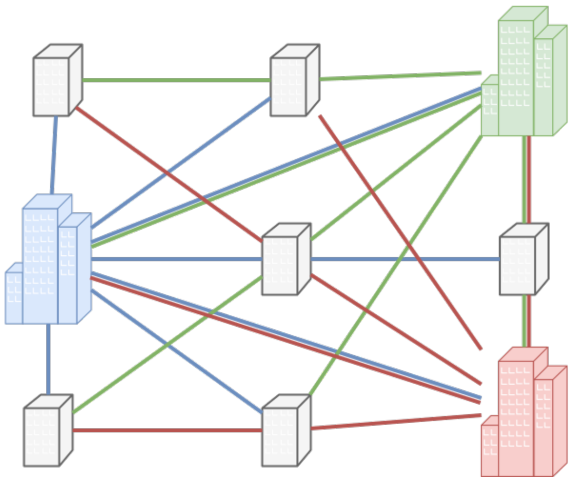


Figure 1.1: Illustration of current logistics networks.

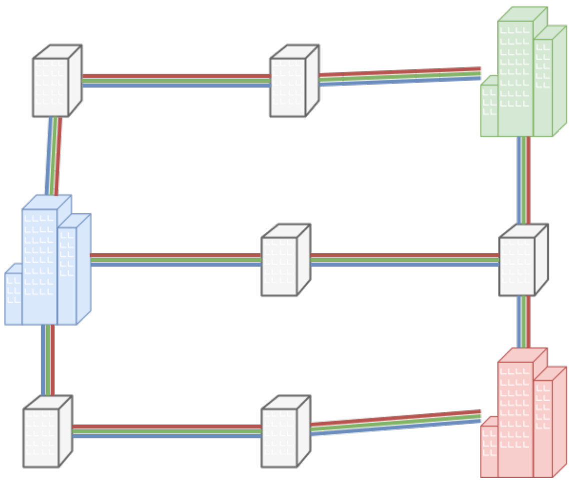


Figure 1.2: Illustration of PI-enabled logistics networks.

Each companies' goods are encapsulated in standardized containers, which are then routed throughout the network like packets in the Internet. At each hub, shipments are taken over by different trucks or switched to other modes of transport. This approach promotes the consolidation of different companies' flows, which in turn should increase the average load fraction in transport vehicles. This change is illustrated in figure 1.3.

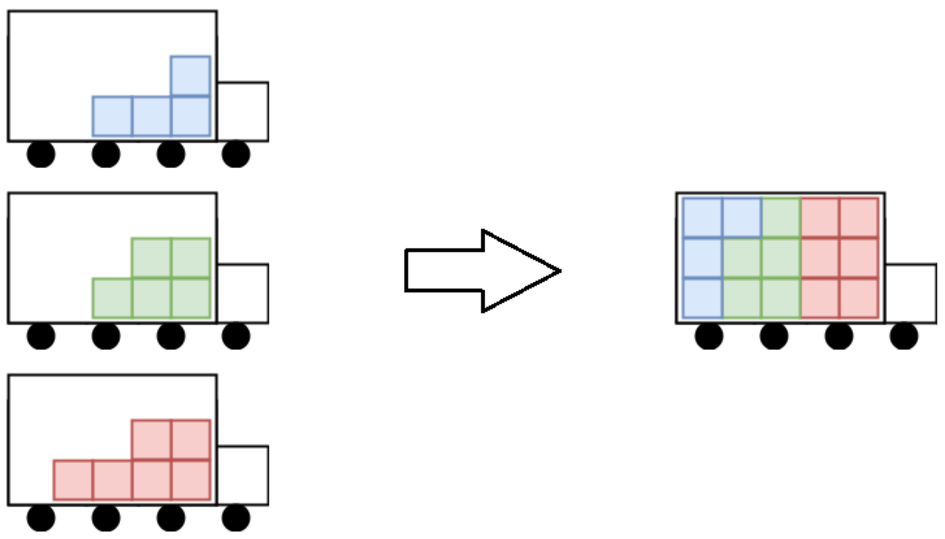


Figure 1.3: Illustration of order consolidation in PI networks.

Like the Internet, all decision-making should be performed by autonomous protocols. These can be responsible for different aspects of the network. In the PI, the major categories are:

- Containerization protocols that decide which goods to put in the same container.
- Routing protocols that choose which neighboring hub a container should go to next.
- Consolidation protocols that assign a vehicle to the containers and dispatches it in time.

Moving to this open global logistics system should promote accessibility and fairness for small companies, who can enjoy the same scale advantages as large corporations [3]. [2] estimates that implementing PI should double the efficiency of current logistics systems.

1.3 Objectives of this dissertation

The main objective of this dissertation is to develop a modular, open-source simulation framework that allows research into routing and consolidation protocols in the PI. More specifically, it should enable researchers to quantify the value of collaboration and information-sharing by simulating increasingly complex protocols.

The secondary objective is to kick-start the development of these protocols. This research will focus on designing fast, rule-based approaches that use little to no information about the other nodes in the network.

Furthermore, the source code for this framework is made freely available on GitHub. This is done to encourage further collaboration between PI researchers and to promote the sharing and development of open-source software.

1.4 Outline

This master's dissertation consists of 6 chapters:

1. The first chapter introduces the PI vision and covers the goal and structure of this dissertation.
2. The second chapter reviews existing literature and identifies gaps in the current research.
3. Chapter three first details the objectives and principles of the proposed framework. It then describes its PI model and explains how it is simulated. This chapter ends by reviewing each assumption and analyzing their impact.
4. The fourth chapter documents three major optimizations that were made during the implementation process.
5. Chapter five demonstrates the versatility of the simulation framework by using it to implement and compare several routing and consolidation protocols.
6. Chapter six ends this dissertation by formulating the conclusions and listing potential future research topics in the field of PI.

Chapter 2

Literature review

This chapter examines the existing literature on the PI. Firstly, it explores the origins of PI research and the reason for its deployment. Secondly, it presents the components that make up a PI-enabled network and takes a closer look at autonomous protocols. This chapter then compares approaches to model and simulate PI networks and summarizes existing simulation research. Afterwards, it checks how PI is being received by industry and identifies possible bottlenecks in executing the vision. The literature review concludes by explaining which research gap this dissertation aims to fill.

2.1 Conceptualization of the Physical Internet vision

The term 'Physical Internet' first appeared on the cover of The Economist in 2006 [4]. This term was not further mentioned inside the newspaper, but it piqued the interest of Professor Benoit Montreuil (Québec, Canada), who began filling out different aspects of the vision. He was joined by Professors Éric Ballot (Paris, France) and Russell Meller (Arkansas, U.S.A.) in 2009. Together, they published the first PI paper outlining the different constituents of a PI-enabled logistics network [5]. The current consensus on how a PI network should function is presented in section 2.2.

From 2009 to 2012, Montreuil iterated on a slideshow titled *Physical Internet Manifesto: Globally Transforming the way physical objects are handled, moved, stored, realized, supplied, designed and used* [3]. This used to be hosted at www.physicalinternetinitiative.org, but the domain has since been taken over by another organisation. This PI manifesto was compiled into a paper in 2011 [6].

Montreuil [6] motivates the need for a new era of logistics by stating the hypothesis that "the way physical objects are moved, stored, realized, supplied and used throughout the world is economically, environmentally and socially inefficient and unsustainable". He elaborates this by providing a list of 13 symptoms, shown in table 2.1.

Table 2.1: Inefficiency symptoms of current logistics. Adapted from [3]

	Inefficiency and unsustainability symptoms	**Economical**	**Environmental**	**Social**
1	Limited fill rate of transportation means	x	x	
2	Empty travel is the norm rather than the exception	x	x	
3	Poor quality of life for truck drivers	x		x
4	Products often sit idle	x		x
5	Production and storage facilities are poorly used	x	x	
6	Many products are never sold or used	x	x	x
7	Products do not reach those who need them the most	x		x
8	Products unnecessarily move, crisscrossing the world	x	x	
9	Fast & reliable multimodal transport is a dream	x	x	x
10	Getting products in and out of cities is a nightmare	x	x	x
11	Logistics networks & supply chains are neither secure nor robust	x		x
12	Smart automation & technology are hardly used	x		x
13	Limited innovation, only local optimisation	x	x	x

2.2 Components of a PI-enabled logistics network

Montreuil presents the PI as a solution to these symptoms. He describes it as "an open global logistics system founded on physical, digital, and operational interconnectivity, through encapsulation, interfaces and protocols" [3].

PI strives for a world where the current proprietary logistics networks are interwoven. The 'Internet' metaphor is used, because it succeeded in connecting countless networks by defining a set of standards.

Arguably the most important standard that enabled this change was the TCP/IP protocol. It encapsulates data by adding layers on top to form a packet. Any packet, no matter which data it contains, can now be routed from hub to hub in the same way by only using these outside layers.

The first major staple of PI is the encapsulation of physical goods in a set of standardized π-containers. In PI literature, standards are often denominated by adding the symbol π as a prefix. Standardizing the dimensions of these containers has many advantages, ranging from increasing the fill rate of trucks to reducing the handling costs for all stages of the delivery. They can be outfitted with a smart tag that functions as the outer layers of the TCP/IP protocol. This enables them to be routed regardless of their content [5].

While PI is mostly being developed for the transportation of goods, the PI manifesto mentions that it could also be used for the transportation of people. By placing them into carts with π-container dimensions, human mobility could benefit from the same sustainability improvements [6]. Research exploring PI for shared public transport has recently started popping up [7].

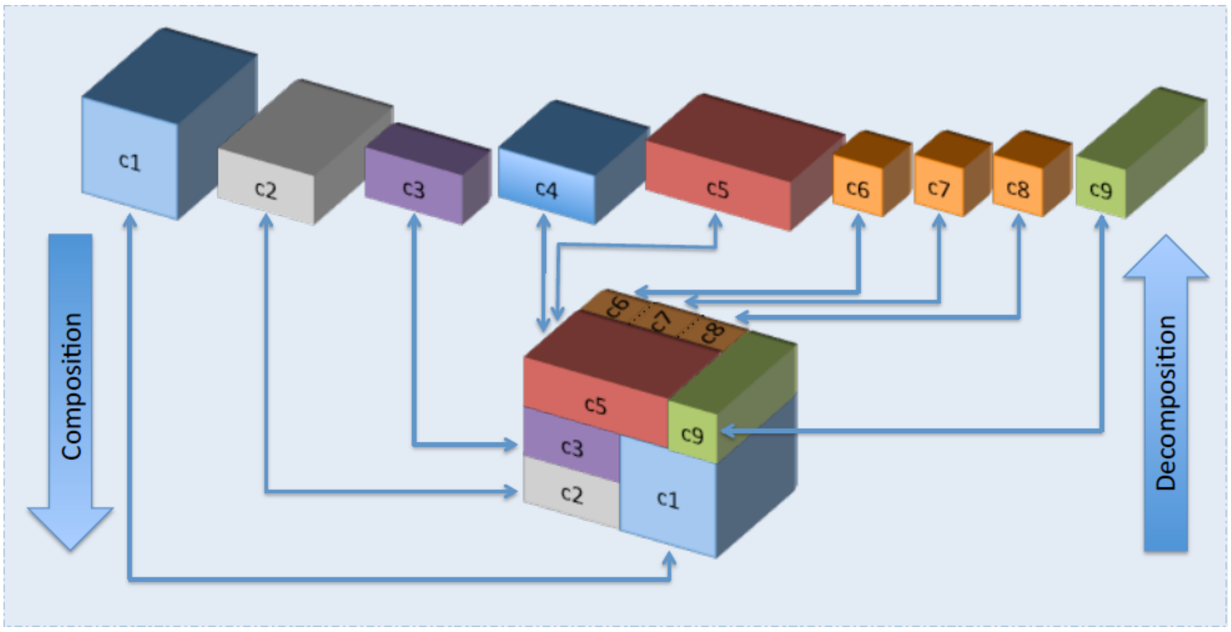


Figure 2.1: Illustrating the modularity of unitary and composite π-containers. [6]

The second pillar of the PI are smart interfaces. These encompass all π-movers (standardized trucks, conveyors, etc.) and π-hubs, which enable efficient unimodal and multimodal transfer of π-containers between π-movers [8]. Similar to the Internet, goods in the PI move from hub to hub on their way to their destination.

The final pillar is the use of standard collaborative protocols [8]. These protocols autonomously make all decisions in a PI network. They should also ensure that all necessary stakeholders receive information about the state of the delivery. The use of autonomous protocols should make a PI-enabled network more robust than the traditional supply chain, as they can automatically respond to failures or bottlenecks in the network. These protocols are at the heart of this master's dissertation.

2.2.1 PI protocols

Using autonomous protocols in the PI is inspired by the way protocols constitute the Internet today. Nevertheless, PI protocols have some fundamental differences: (1) Unlike the links in the Internet, the connections between π-movers do not have a fixed bandwidth. Especially in road transport, the number of π-movers can be adjusted to better match the behavior of the network [9]. (2) In the Internet, a packet is destroyed and re-transmitted when it experiences too much delay. This is impossible in the PI, as π-containers are physical objects.

These differences have not stopped researchers from trying to map the standards of the PI to those of the Internet. [10] adapted the Internet's Open Systems Interconnection model (OSI) to define standards in the PI. This model is called the Open Logistics Interconnection (OLI) model and contains 7 layers, which are depicted in figure 2.2.

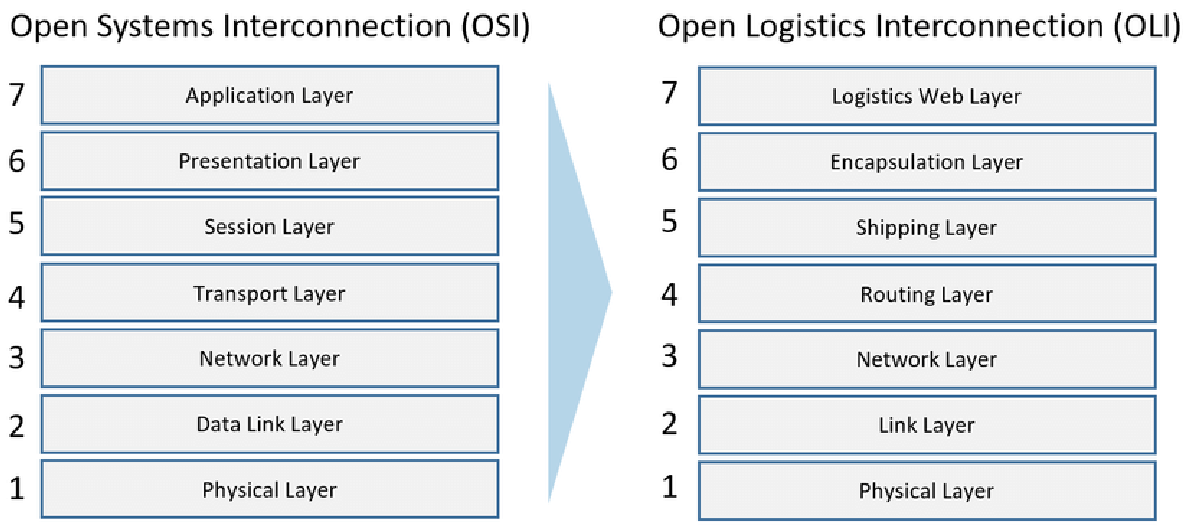


Figure 2.2: Open Logistics Interconnection model. [11]

The framework in this dissertation focuses on designing and analyzing protocols in the 4th and 3rd layers. The 4th is the 'Routing Layer'. It focuses on routing π-containers between π-hubs in the fastest or most cost-effective way. The routing protocol needs to be highly dynamic and respond to changes in the network. The 3rd layer, named the 'Network Layer', is responsible for dispatching. It autonomously assigns π-movers to π-containers and decides when to ship these out.

2.2.2 Modeling and simulation of PI networks

Modeling and simulation has supported researchers and companies in their decision making for a long time. Enabling them to quickly test hypotheses has proven to be extremely beneficial. The domain of operations research has used this incredible technique in the design of many new systems.

Three modeling approaches are currently in use: system dynamics (SD), discrete event simulation (DES) and agent-based simulation (ABS). Agent-based approaches are best suited for modeling complex systems with lots of interacting processes [12]. It enables the researcher to make each agent think and act on a local level and see how this influences the performance of the system on a global level.

For this reason, ABS has been the preferred approach for most PI simulation research. The first large-scale simulation project was performed by Ballot, Montreuil, Glardon, et al. [13] in 2012. This research focused on optimizing the transport of fast-moving consumer goods (FMCG) in France. First, a PI network was designed using a heuristic method. Next, an agent-based model was conceived. Each PI protocol is represented by an agent: (1) The containerization agent takes care of the 5th layer of the OLI model. It bundles goods with similar characteristics and chooses which π-container to put them in. (2) The routing agent calculates shortest paths throughout the network. (3) The consolidation agent loads and dispatches the π-movers. Finally, this model is simulated using historical order data and its performance is compared to current logistics. The protocols in this simulation are simple, they do not cooperate with other agents in the network.

The other side of the cooperation spectrum has also been researched. [14] proposes a PI adaptation of the Internet's Border Gateway Protocol (BGP). It is simulated in the same FMCG network and achieves a better performance than independently optimized protocols. However, this protocol requires a constant exchange of information between network operators. In the current ecosystem, this widespread trust between logistics providers is nowhere to be found.

2.3 Implementation roadmap

Montreuil [3] argues that widespread development and deployment of the PI will not be achieved overnight. Instead, it will be progressively deployed and enhanced by finding new ways to interact between network operators. This was recently corroborated by the European Technology Platform ALICE which has made PI its primary vision to achieve zero emissions by 2050 [2]. They have developed a roadmap for the gradual shift towards PI, which is shown in appendix A.

Validation research (e.g. case studies and living labs) is still in its early infancy [15]. However, interest from industry is increasing and several projects have been completed or are well under way.

The first of them was *Modular Logistics Units in Shared Co-modal Networks* (Modulushca). This project was funded by the European Commission and ran from October 2012 to January 2016. It pooled expertise from research and industry to develop a standardized set of π-containers for the fast-moving consumer goods sector [16].

ICONET was another major initiative that was funded by the European Commission between September 2018 and February 2021 [17]. It expanded the state of the art research on using PI and Internet of Things (IoT) for shared warehousing and e-commerce fulfilment. This was done in four 'living labs' [18]–[21] by collaborating with commercial partners like Procter&Gamble and the Port of Antwerp [22].

ICONET [21] highlights that sharing data and operations is the most important barrier for these open, collaborative logistics networks. Implementing PI will require pushing past a 'Chicken-and-egg' problem of needing shared data to collaborate and achieve efficiency improvements, but needing to prove the cost reductions to incentivize competitors to share their data.

This barrier is also remarked by Danon [23] from Ghent University, who interviewed multiple industry experts. Several pointed out the need for clear definitions of trust relations in a PI network and asked what concrete advantages PI could bring in terms of efficiency improvements for their business.

2.4 Research gap

There currently exists little research that compares the performance of different PI protocols with varying degrees of trust and collaboration. This master's dissertation aims to help address the 'Chicken-and-egg' problem by providing a toolbox that allows companies to simulate how their own operations would perform in a PI setting. It can quantify the exact value of sharing specific information, which will be paramount in convincing stakeholders to invest in the PI.

This chapter has reviewed existing PI literature and summarized its vision for the future. It explored state-of-the-art simulations and then identified gaps in the current research that are holding back widespread adoption.

Chapter 3

Design and implementation of the simulation framework

This chapter starts by translating the previously identified research gap into actionable targets. It then lays out the design principles that will guide the development of the framework. Afterwards, it describes the different entities that make up the PI network model and how these will interact. The next section explains how the simulation is implemented and specifies the procedure for generating random container arrivals. The chapter ends by listing all assumptions and analyzing their potential impact.

3.1 Notation

This dissertation uses the following notation to indicate the nature of an element.

- *Cursive*: variables, attributes and parameters
- **Bold**: classes, methods, functions and filenames

Furthermore, the prefix π is no longer written from this point on. The reader can assume that each object refers to the π-standardized version of it.

3.2 Objectives

The main objective of this dissertation is to develop a simulation framework that allows research into routing and consolidation protocols in the PI. It has to enable researchers to:

1. Easily initialize their testing network and configure the network parameters.
2. Quickly iterate and implement protocols that query the network state in order to make informed decisions.
3. Evaluate the performance of their protocol by providing proper data collection and visualization options.
4. Further develop the simulation to add custom features, optimize its performance or change the behaviour of certain elements.

3.3 Principles

To achieve this goal, the simulation framework was designed with the following four principles in mind:

- Usability

The first principle focuses on making the software easy to comprehend and use. The variables should be properly named and each simulation element needs to have adequate documentation. A folder with example input/output files should also help new users understand the requested format.

- Modularity

The simulation should be divided into completely separate modules in such a way that they are are independent and are able to communicate with each other by calling standardized interfaces. A modular software allows the users to swap out modules, edit them or add new ones.

- Customizability

As mentioned in the previous principle, users should be able to change the simulation by editing modules. Therefore, the standardized interfaces should be as broad as possible. They should provide enough information to allow the development of complex modules (e.g. new protocols) without having to edit all other modules/interfaces.

- Scalability

Additionally, the simulation should be widely applicable to all kinds of testing networks. When adding new components, their time complexity should be analysed to identify and resolve potential bottlenecks. Running a large scale test shouldn't take too much time to allow for quick iteration.

These principles serve as a north star during the development of the simulation and guide multiple decisions. They will be referred to extensively throughout this dissertation.

3.4 Structure of the model

This section provides a detailed overview of the way a PI network is modeled and simulated in this framework. The different elements and their interactions are described and all design choices are motivated.

3.4.1 State Graph

Figure 3.1 shows the State Graph. This graph represents the PI network that is currently modeled and enables the user to easily query state information. It is automatically generated from input files. First, the NetworkX library is used to generate the topology of the network, including the weights of the edges. In this case, these weights represent the driving time between nodes. Afterwards, each node receives two attributes: the attribute *node* contains the corresponding instance of the **Node** class, while the *trucks* attribute contains a list of instances of the **Truck** class that belong to this parent node.

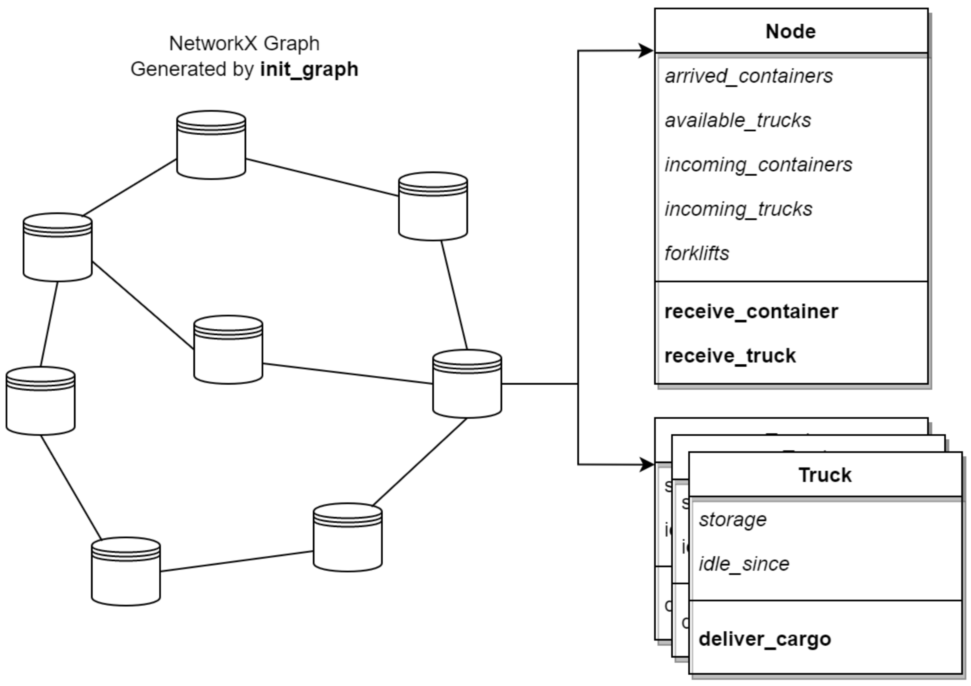


Figure 3.1: Overview of simulation elements: State Graph.

3.4.2 Agents

The **Truck** and **Node** classes are the agents that will interact in the model. In order to keep the framework as modular and customizable as possible, the complexity of these agents should be kept to a minimum. A user shouldn't need to edit a class' methods to change the behavior of the model. After all, changes in the interfaces between agents can cascade and cause bugs throughout the whole simulation.

For this reason, the interactions between agents are kept as simple as possible. Their methods represent the smallest changes possible, called *atomic interactions*. This way, the entire scope of complex agent behaviors can be modeled as a chain of atomic interactions, initiated by a decision making unit or DMU (described in section 3.4.4).

Next to executing atomic state changes, agents are also responsible for correctly representing the state information of the network in their attributes. Together, they form a central database (the State Graph from figure 3.1) that can be queried by different DMUs.

A qualitative description of the agents' attributes and methods is provided.

Class Truck

- Attributes
  - *storage*: containers that are currently inside the truck.
  - *idle_since*: timestamp at which the truck was made available at its current node.
- Methods
  - **deliver_cargo**: this method is called with a cargo and destination as variables. The truck first requests and uses a forklift to load its cargo. Then, it drives to the destination. Once arrived, it again uses a forklift to unload its containers and finally makes itself available at the destination node.

Class Node

- Attributes
  - *arrived_containers*: containers that have arrived at the node. For performance reasons, the protocols in this dissertation move these containers into their own efficient data structure at every iteration. This is discussed in section 4.1.
  - *available_trucks*: trucks that are idle at this node.
  - *incoming_containers*: future arrival times of newly generated containers, e.g. deliveries starting at the current node. All future arrivals are known up to at least a *lookahead* time later, which is a parameter of the simulation.
  - *incoming_trucks*: arrival times of all incoming trucks.
  - *forklifts*: a finite resource that is requested by trucks to load/unload their cargo.
- Methods
  - **receive_container**: this method is called for every incoming container. It edits some of its analytics (e.g. number of hops) and saves them if the container has arrived at its final destination. Otherwise, the container is added to *arrived_containers*.
  - **receive_truck**: this method removes every arrived truck from *incoming_trucks*, resets its *idle_since* attribute and adds the truck to *available_trucks*.

3.4.3 Containers

The agents in this model move around standardized containers. The code implementation of these containers is kept very simple to decrease the run time of the simulation, as they will be moved around a lot. The containers are represented by a tuple of the form **tuple(***due_date*, [*metrics*]**)**. The due date can be used as a priority metric in multiple data structures and algorithms, a selection of these are discussed in chapter 5. The metrics list is updated throughout the delivery journey and is saved as output (see section 3.5.2).

3.4.4 Decision making unit

The decision making unit (DMU) controls the flow of the model and therefore represents the behavior of the agents. When the user wants to test a different PI system, they should only have to edit how the DMU initiates atomic interactions.

This increases the usability of the code base by providing a simple toolbox of atomic class methods. It should be noted that splitting up processes into chains of interactions can increase the time complexity of the simulation.

The DMU can be seen as a layer built on top of this toolbox. It can take many different shapes and make decisions on many levels: from doing network-wide optimization using integer programming to taking fast, rule-based decision at each node.

This master's dissertation focuses on building these relatively simple, rule-based processes. The DMU in this dissertation consists of a routing and consolidation protocol at each node. This protocol decides which containers are loaded onto a truck (consolidation) and to which neighboring node these are sent (routing).

Figure 3.2 shows the basic functioning of such a protocol. At each iteration, it queries the state information by reading some of the node's attributes. Based on these, it can decide to task a truck to deliver a cargo (set of containers) to a neighboring node. It does so by initiating the truck's **deliver_cargo** method, which then calls the destination node's methods to receive the cargo and the truck.

There are multiple ways to decide when to initiate a protocol. These are discussed in section 4.3. In the current implementation of the simulation framework, the protocol is executed at a fixed time interval. The length of this interval is included in the simulation as one of the parameters, discussed in section 3.5.2.

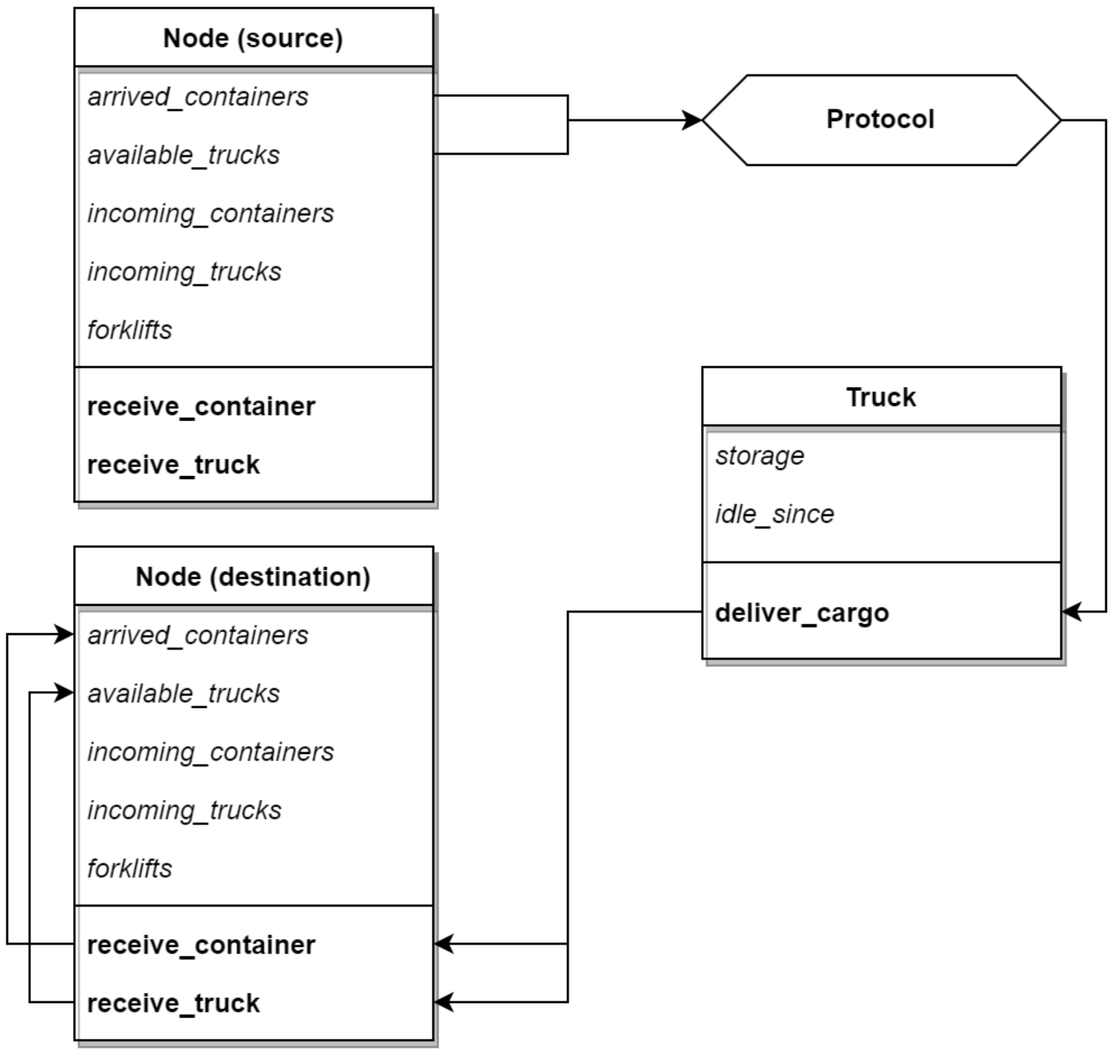


Figure 3.2: Overview of simulation elements: Protocol.

In theory, a developer should be able to implement any decision process using the current agents. In practice however, the time complexity of a DMU can sometimes drastically be reduced by adding a custom attribute to an agent or slightly editing its atomic methods.

The full decision process of several protocols is described in chapter 5.

3.5 Implementation of the simulation

This section discusses how the framework is implemented and specifies the simulation procedure for random container arrivals.

3.5.1 Software

A lot of DES/ABM research projects are built in low-code/no-code toolboxes for which an expensive commercial license is necessary. These programs mostly involve dragging around objects within a graphical user interface (GUI). Some examples of software packages are Simio, Plant Simulation, SimEvents and AnyLogic [24].

These programs enable a relatively fast development cycle for simple projects. However, it is often a struggle to implement complex interactions between objects. This is further hindered by the proprietary nature of these packages. Because they are closed-source, documentation is very limited and community projects are rarely shared. This causes researchers to experience vendor lock-in, due to the steep learning curve these toolboxes have.

Since this framework aims to be very usable and customizable, it is implemented in Python. This language is extremely popular because of its user-friendliness and large range of well-documented libraries. The model is simulated using the SimPy framework. This DES library is also well-documented and can be used to model active agents that interact with shared resources.

The documentation for this framework is embedded in the code base. Each function, class and method contains a docstring and additional comments are added to enhance readability.

3.5.2 Input and Output

As explained in section 3.4.1, the State Graph is generated from input files. The file **nodes.csv** contains the nodes and their characteristics (e.g. number of trucks). The file **edges.csv** contains the existing edges and their weights.

Besides those files, there are two more input files: **orders.csv** contains the instructions for the container generator, explained in section 3.5.3. The file **parameters.csv** contains the parameters of the simulation, which are:

- *sim_days*: Number of days to be simulated.
- *truck_storage_capacity*: Amount of containers that can fit in a truck.
- *handling_time*: Handling time per loading/unloading action of a container. More information can be found in section 3.6.
- *orderfreq_mult*: Multiplier for the arrival frequency of newly generated containers.
- *lookahead*: Size of time interval, starting at current time, in which all future arrivals of containers are known. These arrival times are displayed in a node's *incoming_containers* attribute.
- *protocol_interval*: Time interval between iterations of the protocol.

The framework automatically saves data from the simulation. The current implementation creates two CSV files: (1) **transports.csv** contains information on each truck journey (e.g. departure and arrival time, load, etc.). (2) **deliveries.csv** saves metrics about each container that has arrived at its final destination (e.g. due time and arrival time, total time spent in transport and handling, ...). These files are used to calculate different Key Performance Indicators (KPIs) that are analyzed in chapter 5.

Apart from these raw data files, the user also has the option to generate an in-depth event log, which is saved in **debug.log**. This increases the run time of the simulation, but can be very helpful when debugging.

Example files are provided in Appendix B and C.

3.5.3 Simulating the random arrival of containers

Each replication of the simulation starts with the same initial state: all trucks are idle at their parent nodes and each node has an empty inventory. To start simulating the interactions in a PI-enabled network, containers need to be generated at their starting positions. This is carried out by the Order Generator.

Figure 3.3 shows the functioning of the order generator. It has two responsibilities: (1) generating containers at random times and initiating the **receive_container** method to add them to the system; (2) keeping the *incoming_containers* attribute up to date.

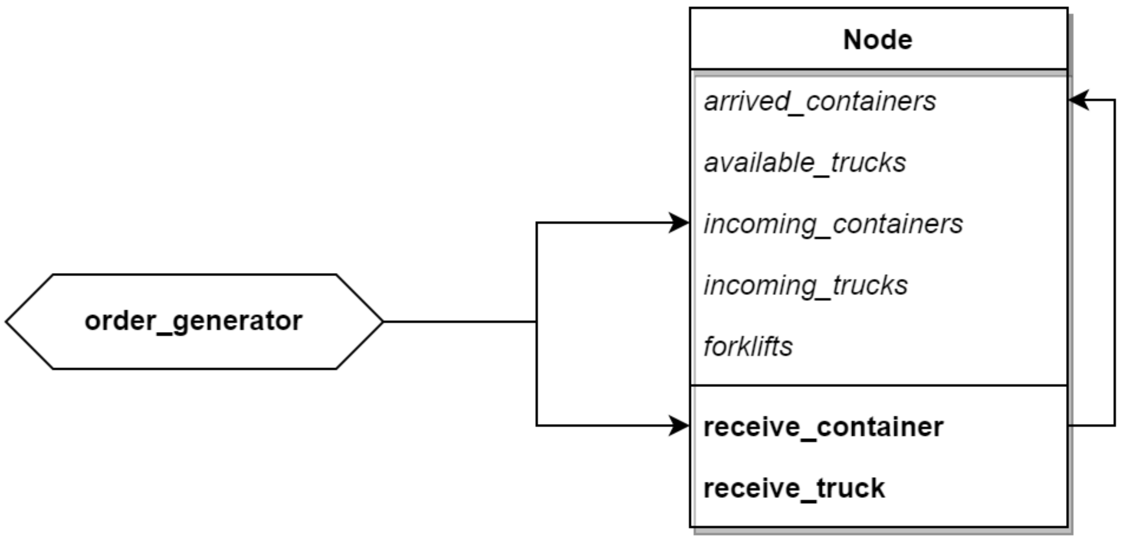


Figure 3.3: Overview of simulation elements: Order Generator.

The input file **orders.csv** details which orders should be generated. Each line contains a *source*, *target* and *interval* value. A generator instance is assigned to each order. It initializes containers at the *source* node and assigns the *target* node as destination. The containers arrive at random intervals, with *interval* being the average interarrival time.

The current implementation of the order generator assumes that the arrival of containers can be described by a Poisson process (see section 3.6). This implies that the interarrival times are independent and follow the exponential distribution:

$$\text{interarrival} \sim \text{Exp}(\lambda)$$

The exponential distribution is defined by its rate parameter λ, which represents the average number of containers that arrive per hour. In this framework, λ is calculated as:

$$\lambda = \frac{orderfreq\_mult}{interval}$$

orderfreq_mult is a universal parameter of the simulation, defined in section 3.5.2. It multiplies the arrival rate of all orders. This is used in chapter 5 to test how the network would perform in more/less busy conditions.

As explained in section 3.4.3, a container is implemented as **tuple(due_date,[metrics])**. The order generator creates the due date by sampling the following uniform distribution:

$$due\_date \sim \text{Uniform}(now + 3 \cdot dist, now + 8 \cdot dist)$$

Where *now* is the current time step in the DES and *dist* the length of the shortest path from the parent node to the container's destination. These boundaries are less biased compared to setting the same goal for each order, as containers that are further from their destination will often need more time to arrive. The distance multipliers are chosen arbitrarily and can easily be edited in the order generator.

As shown in figure 3.3, the order generator is also responsible for correctly populating the *incoming_containers* attribute. For each *target* node, this contains a list with all future arrivals up to at least *lookahead* hours later. *lookahead* is another global parameter of the simulation. The goal of the *incoming_containers* list is to provide certain information that can be used to optimize the routing protocols.

When the simulation is initialized, the order generator fills all *incoming_containers* lists with arrivals. A new arrival is generated by sampling the interarrival time, summing this with the last known arrival and appending the result to the list. This is repeated until the latest container arrives after the *lookahead* period.

The inner workings of the order generator are explained using an example. A possible result of the initialization step is shown on the first axis of figure 3.4. In this example, t represents the current time in the simulation (equal to zero in the first figure). L is the *lookahead* time and the vertical arrows indicate arrivals in an *incoming_containers* list.

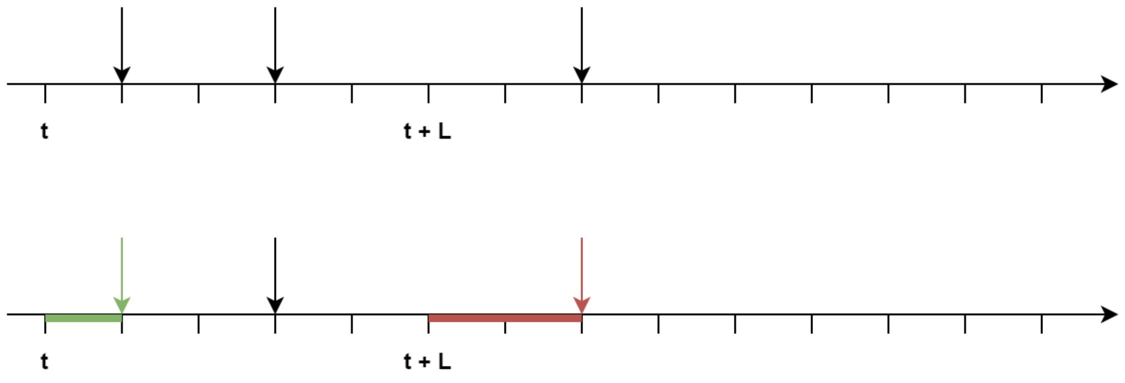


Figure 3.4: Lookahead mechanism example: initialization.

The order generator algorithm keeps track of two time intervals: the interval between the current time and the next container arrival, and the interval between the end of the lookahead period and the last known arrival. The former is indicated in green, while the latter is colored red. Please note that these color codes are used in all figures.

It has to be noted that these figures give the wrong impression that only integer time steps are considered. However, all interarrival times are floating point numbers in the current implementation.

Each iteration, the generator identifies the smallest interval. In figure 3.4, it is the green one. The order generator then uses a **yield** statement to call a timeout for the length of the green interval. This causes the algorithm to be paused until the DES has reached the next arrival. This is represented in figure 3.5 by moving *t*.

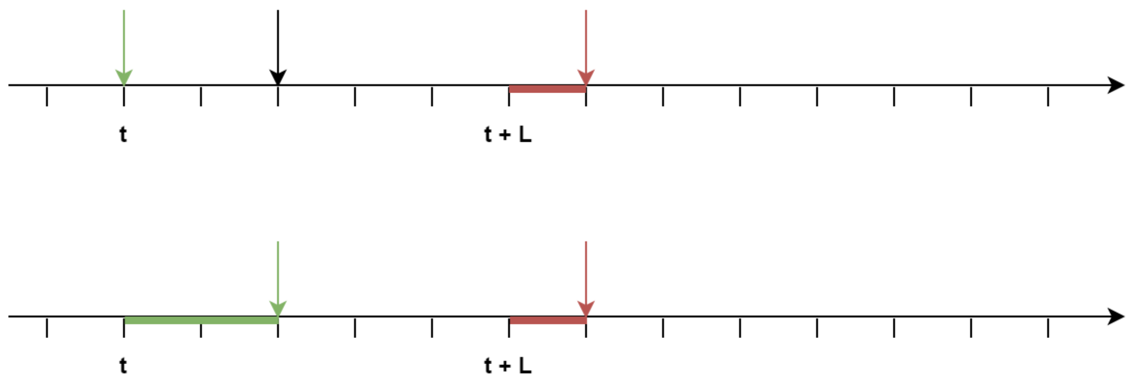


Figure 3.5: Lookahead mechanism example: sending out container.

Once the simulation clock reaches the first arrival, a container is generated and the **receive_container** method of the *source* node is invoked. Afterwards, the first element of the *incoming_containers* list is deleted. On the next iteration, the green interval is recalculated. Notice that the red interval has become the shortest.

This time, the simulation waits until the red interval becomes zero. Once the last known arrival reaches the lookahead time, a new arrival is generated and added to the *incoming_containers* list.

If the latest arrival was at any point inside the lookahead interval, a new arrival could potentially be generated that also falls inside this interval. This violates the goal of the lookahead period, which is to provide a period of certainty that can be used by protocols during routing and consolidation.

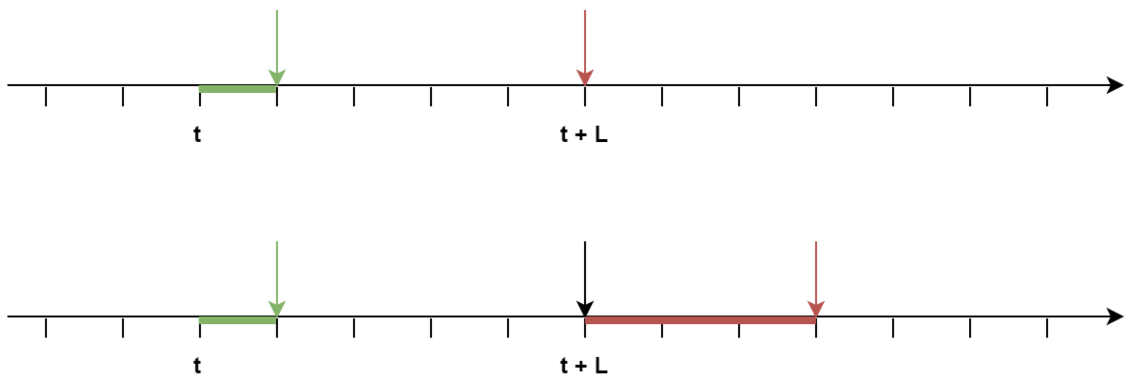


Figure 3.6: Lookahead mechanism example: generating new arrivals.

This process is continued until the end of the simulation. A complete overview of the order generator is provided in figure 3.7. Its implementation was originally not as complex, but was changed to reduce the memory usage of the simulation. This change is discussed in section 4.2.

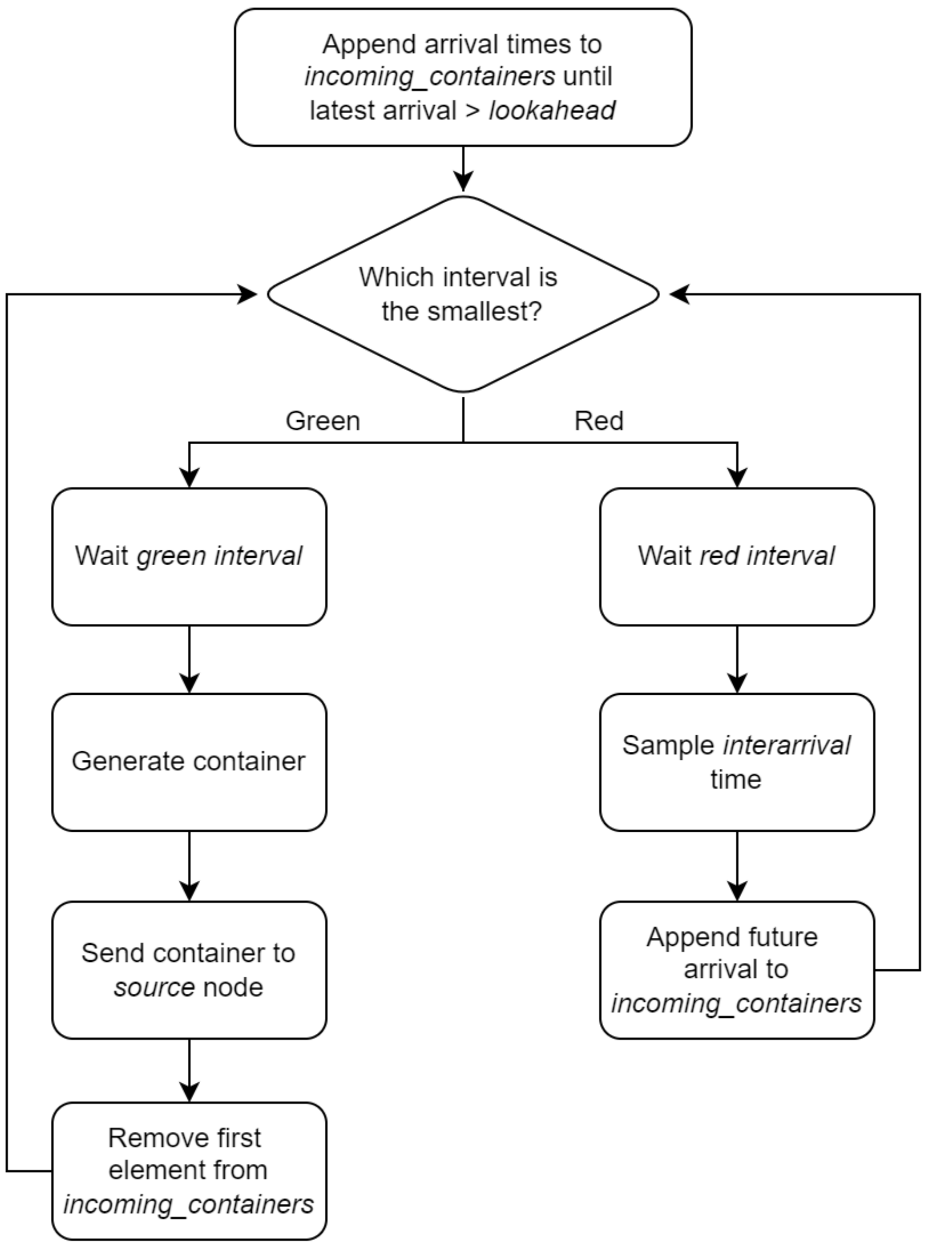

Append arrival times to *incoming_containers* until latest arrival > *lookahead*
Which interval is the smallest?
Green
Red
Wait *green interval*
Generate container
Send container to *source* node
Remove first element from *incoming_containers*
Wait *red interval*
Sample *interarrival* time
Append future arrival to *incoming_containers*

Figure 3.7: Order Generator flowchart.

When the *lookahead* is zero, the order generator does not need to keep track of any future arrivals. This dramatically simplifies its decision process, which is presented in figure 3.8.

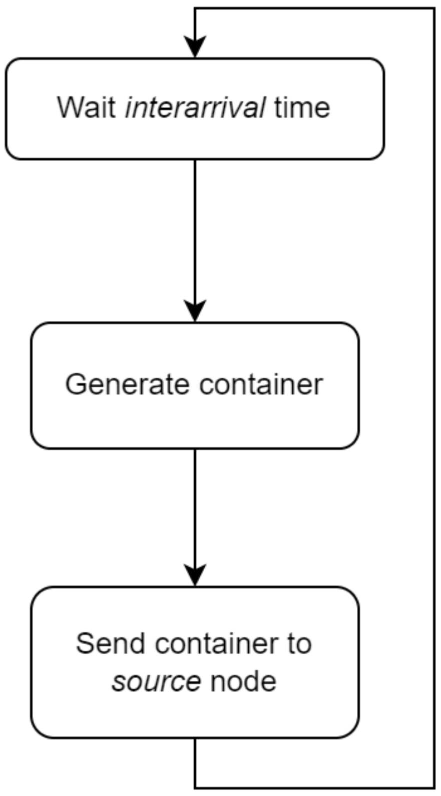


Figure 3.8: Order Generator flowchart when $lookahead = 0$.

3.6 Assumptions

This section lists all the assumptions that were made during the design of the PI network model. The reasoning behind each assumption is first presented, alongside an analysis of how realistic they are. Additionally, their impact on the simulation framework is described.

Assumption 1: Each truck is assigned to a parent node

When initialising the State Graph, each **Truck** agent is assigned to a node in the network. This means that it will only drive on edges that are connected to that parent node.

This assumption follows the vision of the PI, presented in the original manifesto [6]. It states that the PI aims to move from point-to-point deliveries to multi-segment transportation. At each intermediate hub, drivers drop off their containers or simply deposit their trailer. These are then picked up by another driver for the next segment. Meanwhile, the original driver returns to their parent node, potentially carrying cargo with the same destination.

This paradigm ensures that drivers are never far from home. This could potentially help solve the enormous truck driver shortages, which [25] claims are mostly caused by poor working conditions due to long-haul trucking. Assigning a fixed number of trucks per node is thus a realistic assumption. In a real PI setting, this number will be decided and optimized by the node operators themselves.

Due to this assumption, the *available_trucks* attribute of each node contains two lists. The first one has idle trucks that belong to a neighbouring node, while the other contains its own trucks. This distinction is made so the protocol can process these separately.

Assumption 2: The entire network topology is known by each node

In the simulation framework, the weights of the edges in the State Graph (detailed in section 3.4.1) represent the driving times between nodes. Each node knows the entire network topology, including the weights of the edges.

Having access to the network topology is necessary to enable dynamic routing. In the Digital Internet, this is achieved by the Link State algorithm. Link State packets containing the neighboring links and their weights are flooded throughout the network, to make sure each router knows the entire topology. This topology is then used to calculate routing tables. Due to the size and complexity of the Digital Internet, the Open Shortest Path First (OSPF) protocol is used. This divides the global network into local networks connected by a backbone area. Nodes only know the topology inside their local network.

Due to the physical nature of PI networks, their complexity will never come close to that of the Digital Internet. Furthermore, transit and handling times are many orders of magnitude larger. Hence, optimizing the calculation time of routing algorithms is much less important. Therefore, it is realistic to assume that the entire network topology is known to each node. Furthermore, since services like Google Maps and Waze can provide accurate estimates for any journey, it is also realistic to assume that all edge weights are known.

Using their knowledge of the network, nodes can build and continuously update their routing table to tell containers which neighbouring node they should go to next. Knowing the length of this path is also useful, as it provides a lower bound on the delivery time. This will prove especially useful in urgency-based protocols.

Assumption 3: The network topology is constant

This framework also assumes that the edges and their weights do not change during the simulation. This assumption was made to simplify the implementation of the model, as testing the effect of network disturbances would not be researched in this dissertation. It is however not a realistic assumption, as the driving time between places is stochastic. As the topology remains constant, each node only needs to calculate their routing tables once.

Assumption 4: All transport is unimodal

Trucks are presently the only implemented transport vehicle. Other means of transport (trains, airplanes, etc.) are not considered. Therefore, multimodal transport is currently not supported.

The main reason for this is that all examined KPIs are time-based. The implemented protocols thus do not have to make trade-offs between conflicting objectives. They will always take the fastest vehicle available.

This is not realistic according to the PI vision. One of the problems of current logistics, presented in table 2.1, is the lack of fast & reliable multimodal transport. PI promises to deliver this, by implementing multimodal facilities like road-rail hubs. One of its objectives is to make multimodal less-than-truckload shipping as cheap and reliable as single-mode full truckload transport [6].

Assumption 5: All containers are identical

The PI model of this simulation transports containers of one size. The amount of containers that can fit in a truck is given as one of the parameters, which implicitly sets the size of all container to a certain fraction of a truck's storage volume.

This assumption was made to simplify the framework, as the focus is on protocols that route containers to their destination and consolidate multiple flows. Finding the optimal way to pack the containers in a truck was not deemed to be a priority. In reality, a set of standardized container sizes will be used. Their sizes range from entire shipping containers to small boxes. Deciding which set would be optimal has already been extensively researched (e.g. [16], [26]), but a general consensus has not been reached. It is thus not realistic to assume all containers have the same dimensions.

Having multiple container sizes makes it harder to entirely fill a truck. This assumption causes the simulation to overestimate the truck fill rate. However, this overestimation will be consistent across different protocols (assuming they use the same truck packing protocol) and should have no effect on the comparative analysis in chapter 5.

Assumption 6: All containers are fully packed

Containers are seen as the object that has to be routed and delivered. All KPIs are calculated based on the performance of these containers. However, as described in section 2.2, they only act as a receptacle for physical goods. In the calculation of the on-time deliveries fraction, all containers are counted as one delivery. Since they each contribute the same weight, the framework is assuming each container has the same amount of orders inside. Furthermore, truck fill-rate is also calculated using containers. This implicitly assumes each container is fully packed. In the current consensus on how PI should work, containers are only opened at the source and destination node. Their contents remain constant throughout the entire trip, which closely resembles the functioning of the TCP/IP protocol in the Digital Internet. This means that the containerization of goods is a completely separate problem from the routing and consolidation of containers. Therefore, the containerization problem is omitted from the framework as it is not relevant for the currently implemented protocols.

This assumption has a large effect on the absolute values of the KPIs, as the average fill-rate of containers will certainly be lower than 100%. However, this should again not have an effect on the comparative performance of different protocols. According to the PI vision, these protocols do not take into account the content of the containers by design. They only look at the routing information on the outside.

Assumption 7: Containers are an infinite resource

This framework generates new containers at source nodes. When they arrive at their desired destination, these containers are destroyed. Not considering the continuity of containers drastically simplifies the operations in the PI network. This assumption is widely used in PI literature. [9] argues that, due to its complexity, container fleet management should be researched separately. Complex protocols will be needed to make sure that nodes do not run out of containers. In networks with unbalanced demands, trucks will have to transport empty containers.

Assumption 8: Truck drivers and depot workers are available 24/7

The PI networks that are implemented and studied in this dissertation assume that workers are always available. It does not take into account the length of their shift or other limitations.

This was assumed to simplify the framework. Since containers are continuously generated, handled and transported in the same capacity throughout the simulation, the protocols and other agents do not need to change their behavior depending on the time of day. This is not a realistic assumption. First of all, the rate of newly generated containers and the available workforce often depends on the time of day. Secondly, truck drivers work in shifts and should be idle at their parent node in time to return home.

This assumption causes the framework to overestimate the efficiency metrics of the PI. However, this effect should be the same across protocols (assuming they deal with worker constraints the same way) and should therefore again not have any influence on comparative analysis.

Assumption 9: Trucks are serviced in a FIFO queue

Before trucks are loaded/unloaded, they request a forklift from the node they are handled at. This forklift is released after the servicing is complete. The node puts every request in the same First In, First Out (FIFO) queue without regard for the type of request or the contents of the truck.

This approach is assumed to simplify the agent interactions. It is not unrealistic to assume this approach is used at some companies. Nonetheless, formulating a smarter scheduling system can greatly improve the efficiency of the node. Companies often have their own set of rules to decide which trucks should be serviced first.

Assumption 10: New order arrivals follow a Poisson process

As explained in section 3.5.3, the arrivals of newly generated containers follow a Poisson process. The input files contain the mean interarrival time for each order type. Using a Poisson process to model arrivals is a common assumption, because of the 'memoryless' property of exponential distributions. This states that arrivals are independent of each other, which greatly simplifies the simulation. Furthermore, the sum of several Poisson processes is also a Poisson process with an arrival rate that is the sum of the arrival rates of its constituents. This allows the simulation to model different companies' container production using a single generator.

Assumption 11: Container handling time is fixed

The handling time per container is one of the parameters of the simulation, presented in section 3.5.2. It is assumed that the loading/unloading time of a truck only consists of a fixed amount of time per container.

This assumption was added to simplify the **deliver_cargo** method of the **Truck** class. It is however not very realistic, as the total handling time should reflect the time needed to bring a forklift to/from the truck and open/close its storage. This part does not depend on the amount of containers that are present in its cargo.

Assuming a realistic handling time per container is chosen, the current model will underestimate the handling time for partially empty trucks. For filled trucks on the other hand, the handling time will be overestimated.

This chapter started by presenting the objectives and design principles of this simulation framework. It then described the architecture of the PI network model and explained how the simulation functions. The final section showed which assumptions were made and what their impact could be.

Chapter 4

Framework optimization

The fourth chapter documents three major design choices that were made to optimize the scalability and usability of the simulation framework. The first one calculates the most efficient way to store and access the inventory of a node. The second section compares different approaches for generating future arrivals. The chapter ends by explaining why protocols are synchronized across hubs.

4.1 Node inventory bookkeeping

When the protocol dispatches a truck, it first decides what cargo will be loaded onto it. This choice is simple: the containers with the lowest *due_date* take priority. Thus, it needs to find the k containers with the smallest *due_date* values, out of an average node inventory of n containers.

The variable k will never be larger than *truck_storage_capacity*, which sets the maximum amount of containers that can fit in a truck. The node inventory n does not have an upper limit. In a steady state simulation, it should fluctuate around a certain amount. However, when the container volume is too high, the node inventory will build up and become very large.

There are two approaches to find the k smallest values in a list of n elements. The first is to sort the entire list and retrieve the first k values. This has a time complexity of $\mathcal{O}(n \log n)$. The second approach consists of building a min-heap and repeatedly popping the smallest element. This has a time complexity of $\mathcal{O}(n + k \log n)$.

In unstable simulations, n will become much larger than k, which makes the second approach faster. For this reason, heaps are used to store and retrieve containers. However, the framework can not simply put all of a node's containers in a single heap. A truck that is dispatched to a certain neighbor should only carry containers that need to go in that direction. Unfortunately, heaps do not support filtering their elements. Therefore, the inventory should consist of multiple heaps.

The complexity of this inventory structure depends on the features of the protocol. Recall that this framework strives to be customizable and modular. Thus, it is best practice to let the protocol keep track of the inventory and decide which data structure to use. Sections 5.3.1 and 5.4.1 describe two possibilities.

The time complexity of a protocol iteration is further reduced by making nodes maintain an *arrived_containers* attribute. This has the benefit that the protocol's heaps only need to integrate a small subset of containers at every iteration, instead of rebuilding their heaps from scratch. This reduces the complexity of one protocol iteration from $\mathcal{O}(n + k \log n)$ to $\mathcal{O}((c + k) \log n)$, where c is the average number of newly arrived containers. As k is nearly constant, this new attribute reduces the complexity of the protocol from $\mathcal{O}(n)$ to $\mathcal{O}(c \log n)$.

Originally, the simulation framework also maintained the full inventory of a node as one of its attributes. This feature was initially kept to enable querying in the State Graph. As previously explained, this attribute could not already contain a custom data structure. Therefore, its inventory had to be stored as a list.

This attribute was removed because it drastically reduces the scalability of the simulation. Firstly, it uses memory to store redundant information. Each container is stored twice: once in the node's list and once in the protocol's custom structure. Secondly, this double storage forces the protocol to remove a container from the node's inventory every time it is dispatched. This can not happen faster than $\mathcal{O}(n)$, as the node's inventory list is not partitioned by container type. This linear term overpowers the logarithmic time complexity of the protocol.

The difference in run time this linear term makes is shown in figure 4.1. Note that the run time only starts growing significantly when the order frequency multiplier is larger than 2. This happens because the simulation becomes unstable and n starts growing rapidly.

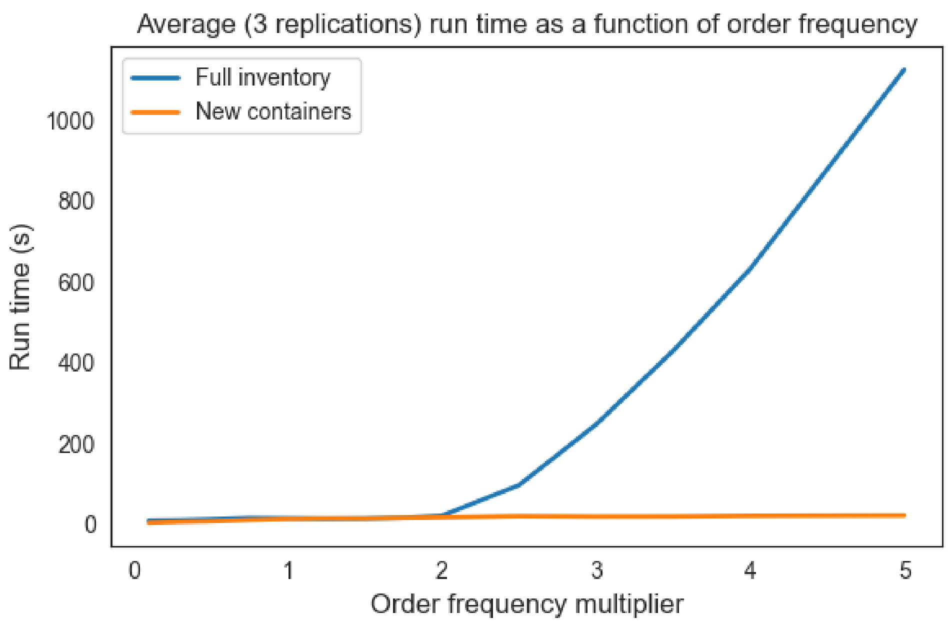


Figure 4.1: Average (3 replications) run time as a function of *orderfreq_mult* (Network: Belgium; Protocol: Volume).

4.2 Order generation

Section 3.5.3 explains the inner workings of the current order generator. Originally, its algorithm was a lot simpler. It would start by generating all arrivals until the end of the simulation and store these in *incoming_containers*. The node's protocols were then responsible for only using information about containers in their *lookahead* period.

This approach had a comparable run time to the current implementation, as shown in figure 4.2. However, it required a lot of memory to store all arrivals and was therefore redesigned.

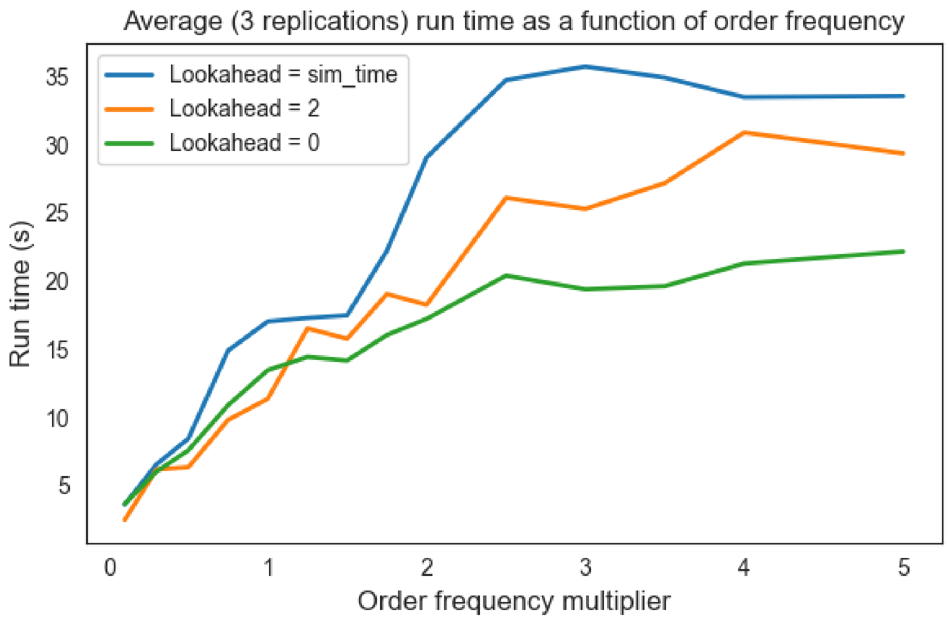


Figure 4.2: Average (3 replications) run time as a function of *orderfreq_mult* (Network: Belgium; Protocol: Volume).

4.3 Protocol synchronization

The final change that happened during the implementation process was migrating from asynchronous to synchronous protocols. Asynchronous protocols are launched every time one of its node's methods is invoked. In other words, they are called when a new container is generated or when a truck is made available.

Initiating a protocol this way has some disadvantages: (1) It will often run without sending out any trucks, as the changes in between replications are minimal. (2) Multiple instances of the protocol can be launched at the same time. This results in a lot of errors because each instance is sending out trucks based on the same inventory signals. Managing this issue proved to be very challenging and significantly reduced the scalability of the simulation. (3) Asynchronous control makes it harder to collaborate with other nodes in large scale optimization efforts.

Because of these disadvantages, the protocols were synchronized. Each of them is now called every fixed time interval. The length of this interval is controlled by the parameter *protocol_interval*.

This chapter documented the three design choices that had the biggest impact on the scalability and usability of the simulation framework.

Chapter 5

Protocol design and evaluation

Now that the PI toolbox has been thoroughly described, the fifth chapter kick-starts protocol research in order to show the versatility of the simulation framework. The first section describes which tools are available to analyze their performance and troubleshoot unintended behavior. Next, a generalized form of a protocol's decision process is presented. Having covered these aspects, the chapter thoroughly investigates rule-based static and dynamic routing protocols. This investigation presents several decision rules and compares their performance in multiple scenarios.

5.1 Evaluation methodology

This framework offers multiple ways to evaluate and understand the performance of routing & consolidation protocols. It also contains diagnostic tools to troubleshoot and detect unintended behavior.

5.1.1 KPIs

As explained in section 3.5.2, the simulation generates multiple output files. This data can be used to calculate several KPIs in order to quantify the performance of the PI network. These KPIs are:

- *On-time delivery fraction*: The percentage of containers that arrive at their destination before their due date.
- *Lateness*: The difference between the actual arrival time of a container and its due date. A positive lateness value means the order arrived late.
- *Total delivery time*: The total time it takes for a container to ship to its destination. This time consists of handling, transport and idle time.
- *Total truck driving time*: The sum of all truck driving times.
- *Average truck load fraction*: The average fraction of a truck's cargo that is filled with containers during transport.

Multiple graphs can be produced to study these KPIs. They will be presented and explained during the analysis of several protocols in sections 5.3 and 5.4.

5.1.2 Warm-up period

The KPIs from the previous section try to quantify how the system performs in an infinite horizon, while only simulating for a limited duration. This is only possible if the system reaches a steady state. Therefore, it is important to check if this is the case. Figure 5.1 shows an example of a network that reaches a steady state after 30 simulated days. These 30 days are the *warm-up period* and are excluded from the calculation of KPIs because they are not representative of how the network usually performs.

Unless stated otherwise, all KPIs and graphs that are displayed in this report have been calculated using data from a network in steady-state, with the appropriate warm-up period excluded.

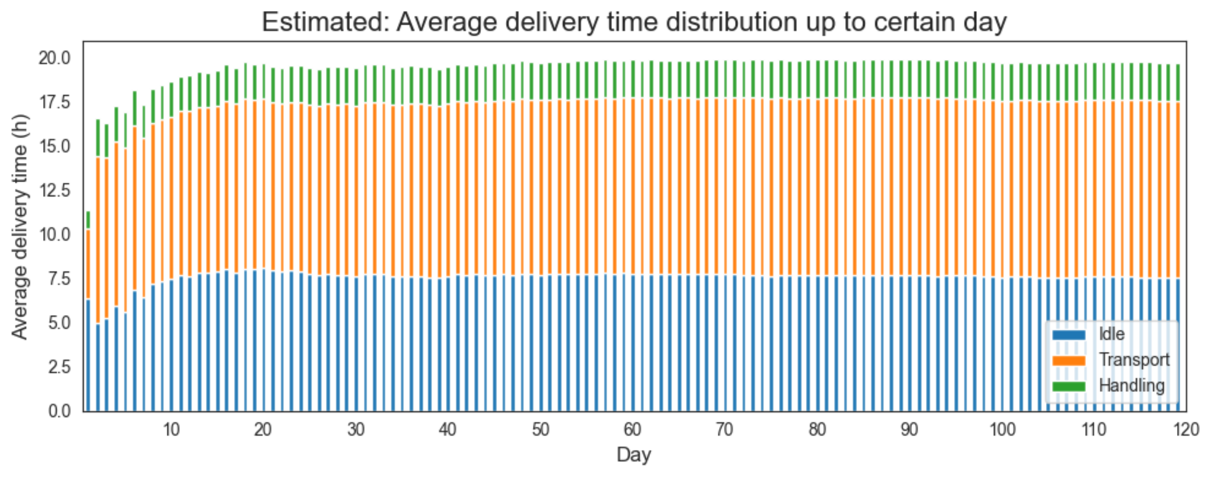


Figure 5.1: Average delivery time evolution that reaches steady state.

If the network is not able to handle the traffic load, an inventory of containers will build up at each node. This buildup might not yet have a large effect on the calculated KPIs due to the limited simulation time. However, on an infinite horizon, containers will have an infinite delivery time. This buildup is shown in figure 5.2.

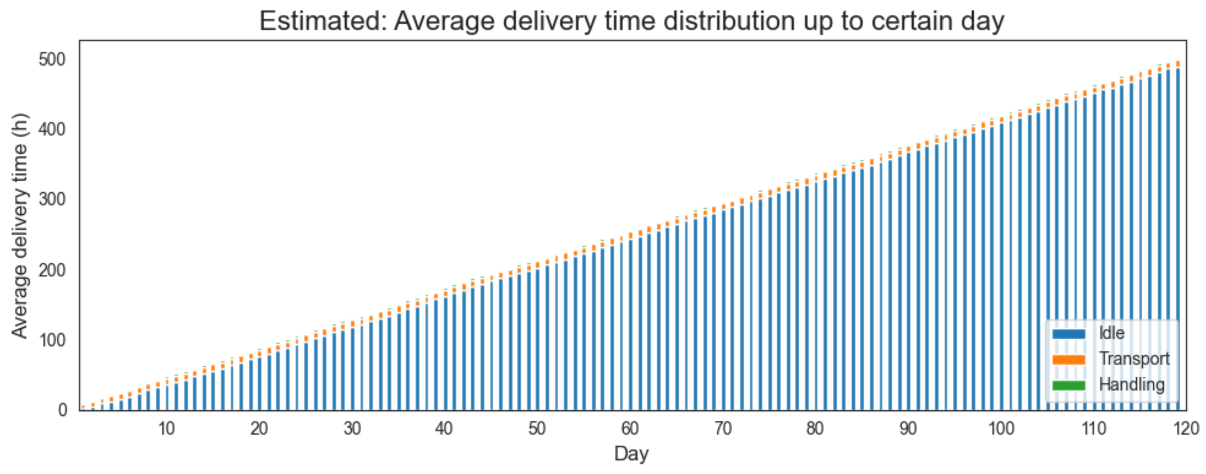
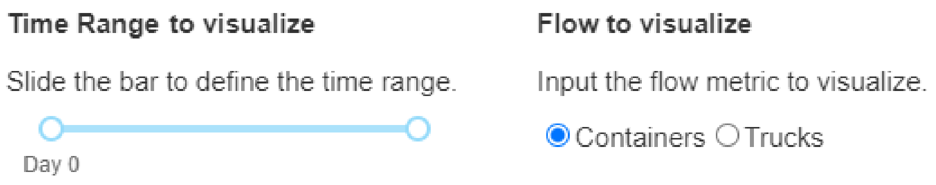


Figure 5.2: Average delivery time evolution that never reaches steady state.

5.1.3 Visualization

Often, the KPIs differ greatly depending on the container type. To understand why these orders deviate from the rest, it is handy to visualize the position of the nodes and the flow throughout the network.

For this, a web application is written in Python using the Dash library. The output files are used to calculate flows between nodes. The visualization can be configured inside the application, as shown in figure 5.3. Researchers can choose which time range to consider and can also pick which metric the calculation uses. Figures 5.4 and 5.5 show an example of each metric.



Time Range to visualize

Slide the bar to define the time range.

Day 0

Flow to visualize

Input the flow metric to visualize.

[✓ Containers] Trucks

Figure 5.3: Web application settings.

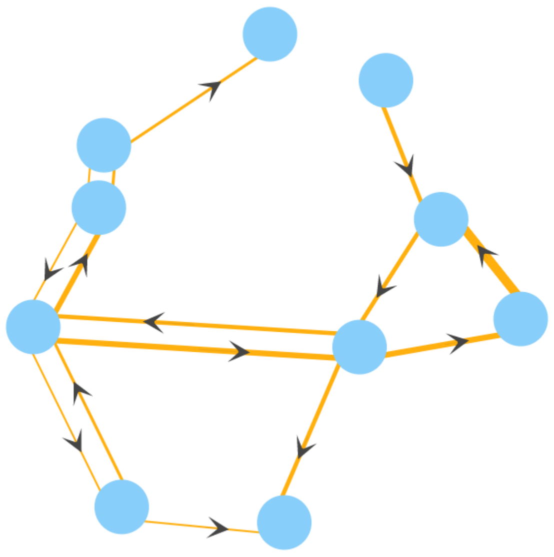


Figure 5.4: Container flow visualization.

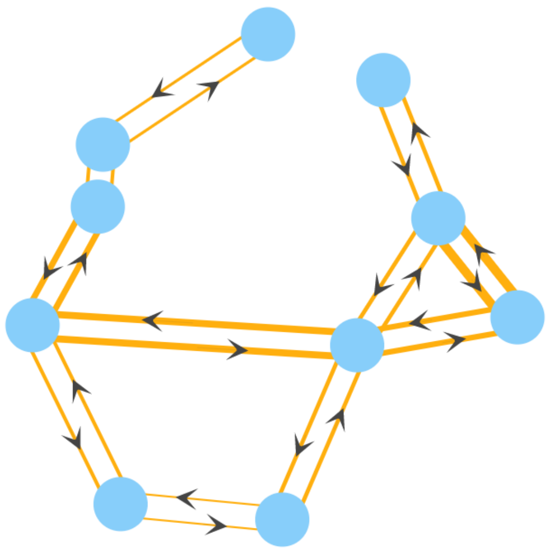
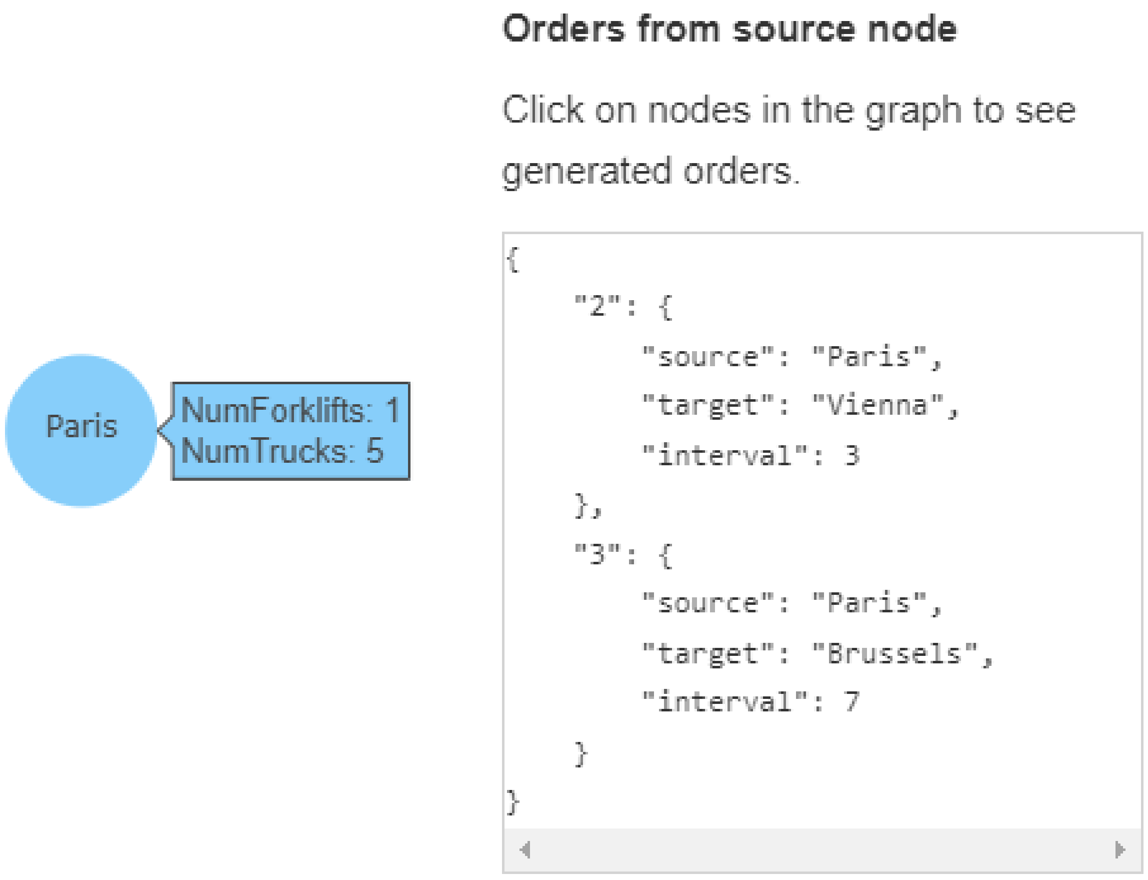


Figure 5.5: Truck flow visualization.

Furthermore, the application also displays important information from the input files. This is illustrated in figure 5.6. When hovering over a node, its allocated resources are revealed. If the user instead clicks on the node, the information pane on the right is populated with that node's order generator input.



Orders from source node

Click on nodes in the graph to see generated orders.

```
{
    "2": {
        "source": "Paris",
        "target": "Vienna",
        "interval": 3
    },
    "3": {
        "source": "Paris",
        "target": "Brussels",
        "interval": 7
    }
}
```

Figure 5.6: Web application information display.

5.1.4 Test networks

Two networks are crafted to use during the testing of protocols. Figure 5.7 shows the small-scale (i.e. the distance between nodes is small) network, which is set in Belgium. The large-scale network spans Europe and is shown in figure 5.8. Their input files and parameters can be found in Appendix B.

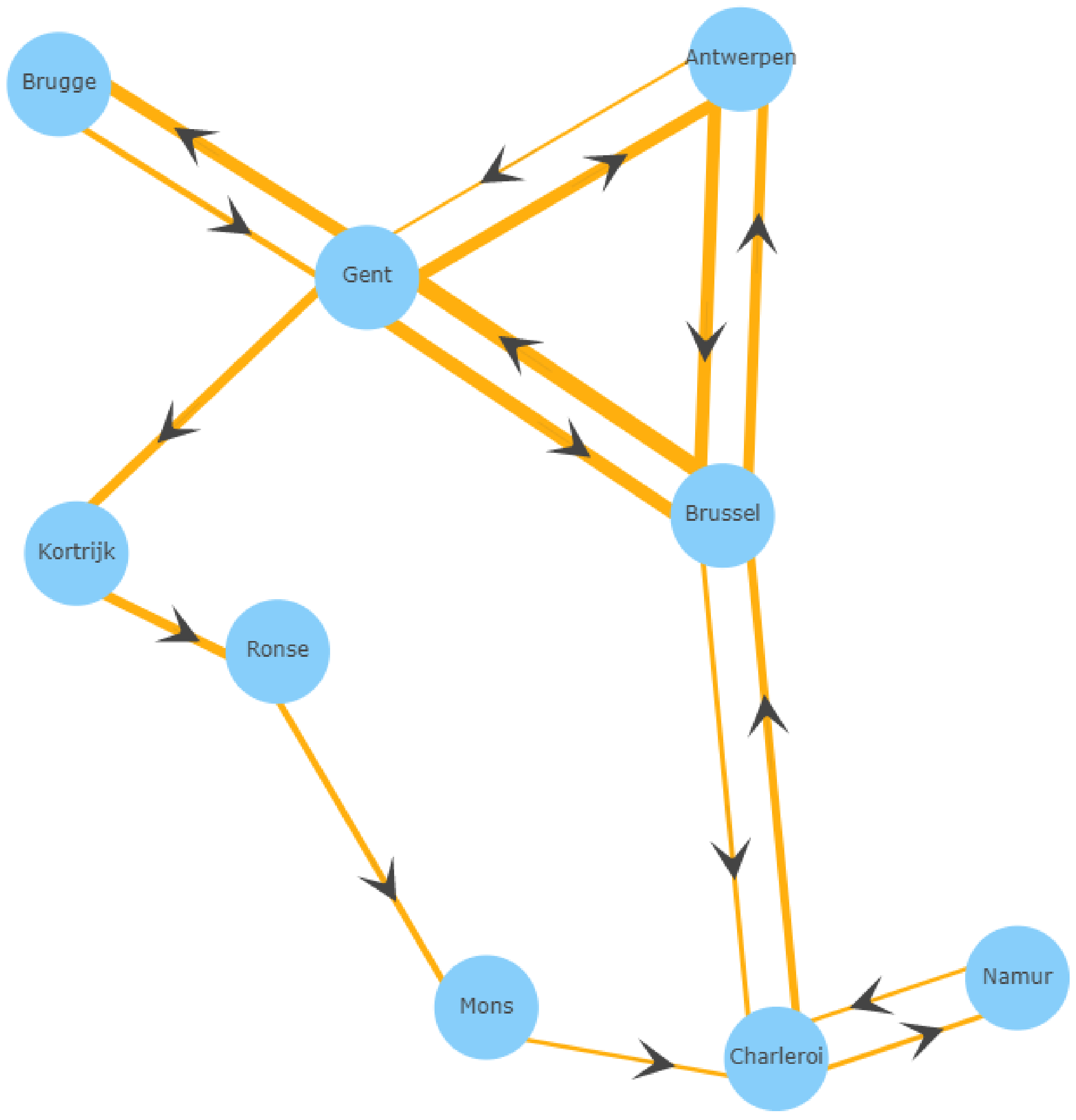


Figure 5.7: Small-scale network: Belgium.

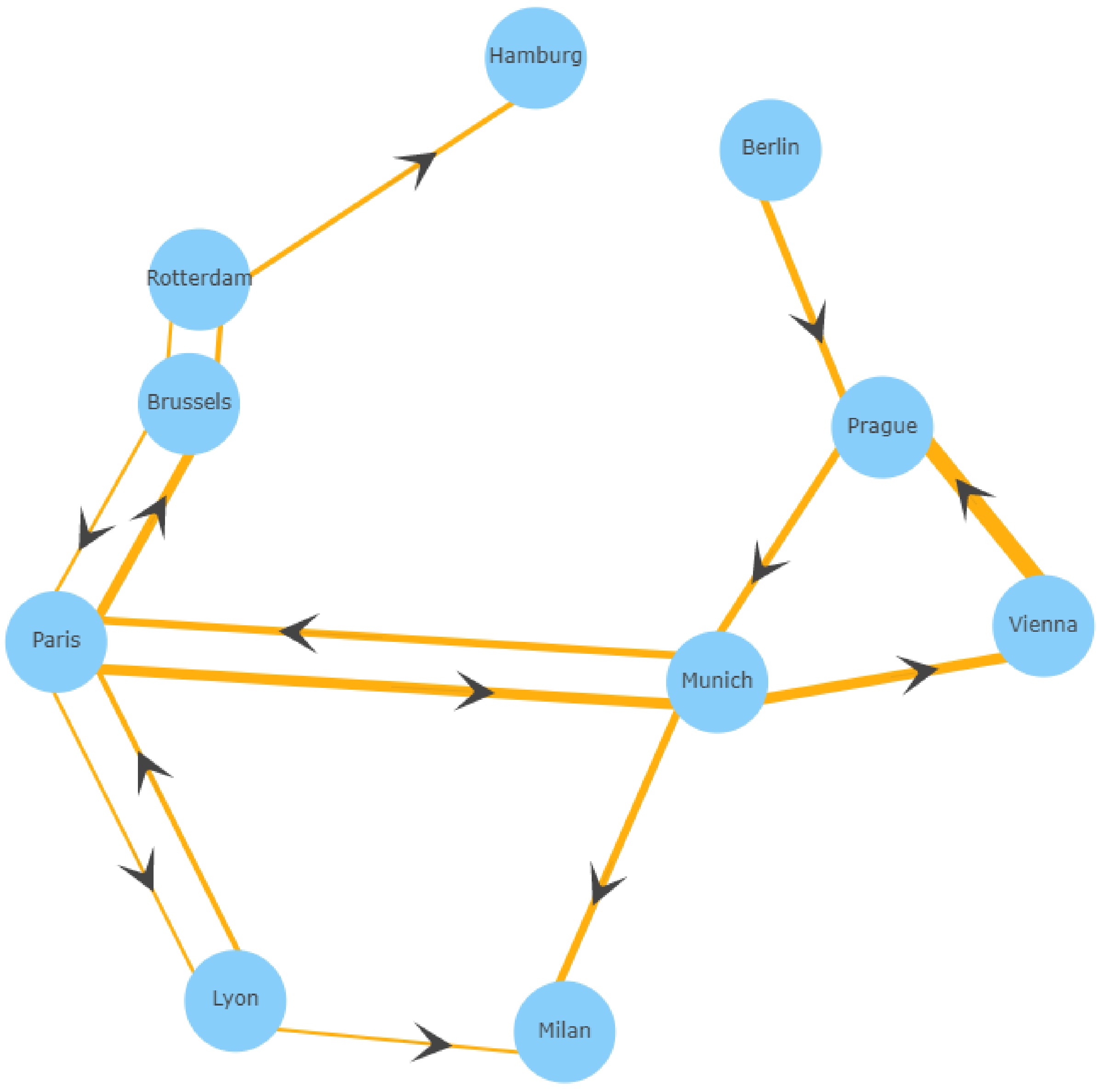


Figure 5.8: Large-scale network: Europe.

5.2 Generalized routing and consolidation protocol

Section 3.4.4 explained how the protocol interacts with other elements in the simulation. This section is dedicated to showing the inner workings of the protocol. Figure 5.9 contains a flowchart to visually represent its internal process.

Each protocol is first activated when the simulation starts. The first step consists of initializing several state variables that are used further on.

In the second step, the protocol updates its internal state by querying the necessary information from the state graph. The protocol is responsible for keeping track of which containers are present at its parent node. Therefore, it is essential that the newly arrived containers are moved from the nodes' *arrived_containers* attribute into the protocol's more efficient data structure. The complexity of this data structure depends on which functionality is needed for the protocol's rules. Two implementations are presented in sections 5.3.1 and 5.4.1.

Once the protocol is up to date, it checks its dispatch rules. If a rule is triggered, a cargo and destination are defined and assigned to a truck using that truck's **deliver_cargo** method. These rules are sorted in order of importance: the protocol only starts checking the next rule when the current one is not triggered.

After checking all rules, the protocol uses a **yield** statement to pause its execution. As explained in section 4.3, this is done to synchronize the protocols across nodes. The length of this pause is defined by the parameter *protocol_interval.*

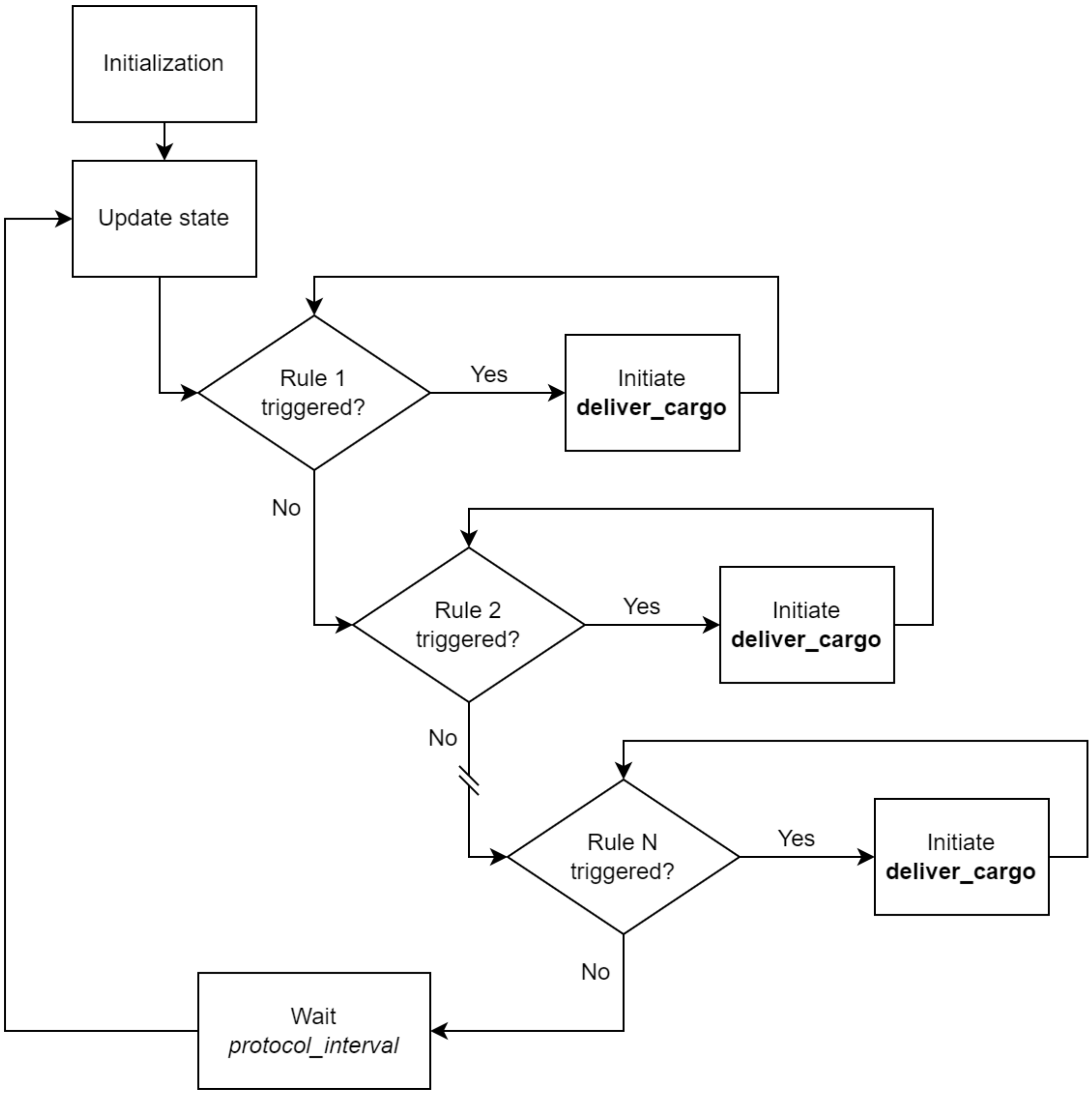


Figure 5.9: Generalized routing and consolidation protocol.

5.3 Static routing protocols

Static routing protocols are first investigated in this dissertation. These protocols do not change their routing decisions during the simulation.

As explained in section 3.6, this framework assumes the network topology and edge weights remain constant. Therefore, the shortest paths throughout the network only need to be calculated once. Each node executes Dijkstra's algorithm to form a routing table.

5.3.1 Inventory structure

Since each container will always be routed to the same neighbor, static routing protocols can group the node's inventory by next destination. As explained in section 4.1, the protocol uses a heap per group to store and retrieve its containers.

5.3.2 Rule 1: Return trucks to their parent node

The first rule in every protocol from this dissertation is the same. It simply checks if the node has any idle trucks that belong to other nodes. If this is the case, these are immediately sent back. Of course, if any container needs to go in that direction, it is loaded onto the returning truck. Prioritizing this rule is important to avoid empty travel and optimize the truck's fill rate.

5.3.3 Rule 2: Dispatch full trucks

The second rule that is discussed is used to dispatch a node's own trucks. It checks if the biggest presorted heap contains enough containers to completely fill a truck's storage. If this is the case, the most urgent items are loaded and the truck is sent to that heap's destination.

5.3.4 Protocol 1: Volume

The first protocol that is tested consists of rule 1 and rule 2. The performance of the PI network using this protocol is used as a baseline to compare with other protocols. Its performance is first investigated on the large-scale network. Figure 5.10 shows the on-time delivery fraction per order. It also includes the 95% confidence interval, which is trivial to calculate as the variable follows a binomial distribution. The average on-time fraction is 99.27[99.03,99.51]%. There seems to be a correlation between the size of the confidence interval and the on-time fraction. This will be explained later in this section.

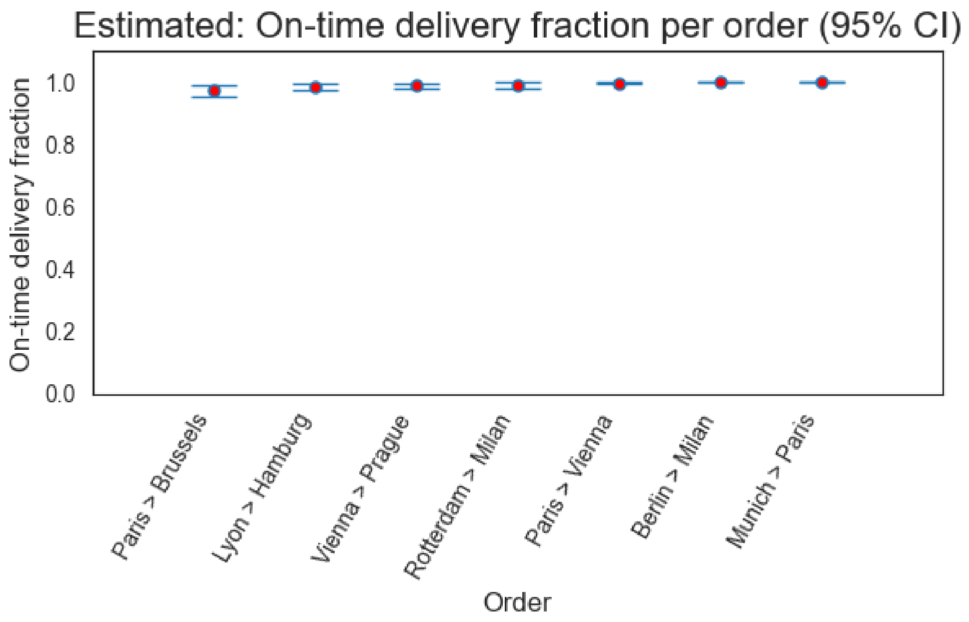


Figure 5.10: Estimated: On-time delivery fraction per order
(Network: Europe; Protocol: Volume; *orderfreq_mult*: 1).

The sample average truck load fraction is 56.31%. The empirical distribution is shown in figure 5.11. The explanation for this distribution is intuitive: every truck is only sent out from its parent node if it can be filled, which is why half of all trucks are full. The rule described in section 5.3.2 then sends these trucks back immediately. This causes most returning trucks to be sent back empty. Sometimes, a couple of containers are available to load before departure.

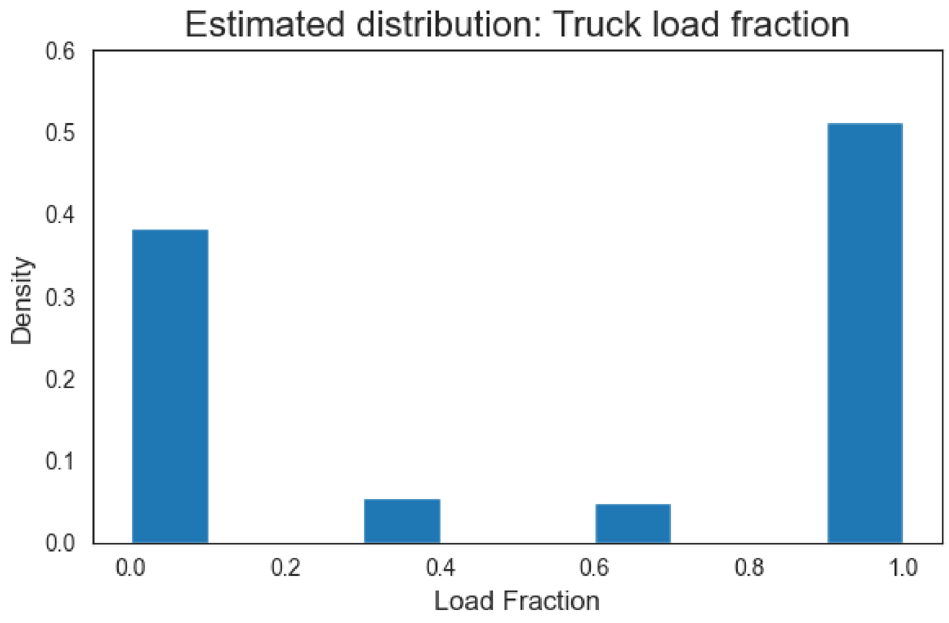


Figure 5.11: Estimated distribution: truck load fraction
(Network: Europe; Protocol: Volume; *orderfreq_mult*: 1).

Figure 5.12 shows what the average total delivery time consists of for each order. This chart shows a clear relation between handling time and total delivery time. However, this is not a direct causal relation. A longer handling time is mostly an indicator of how many intermediary nodes a container has to pass through. When a container visits more nodes on its journey, it has to be dispatched more often and therefore spends more time idle in inventory. This is confirmed by looking at the visualization. The longest orders pass through 3 intermediary nodes, while the shortest directly reach their destination.

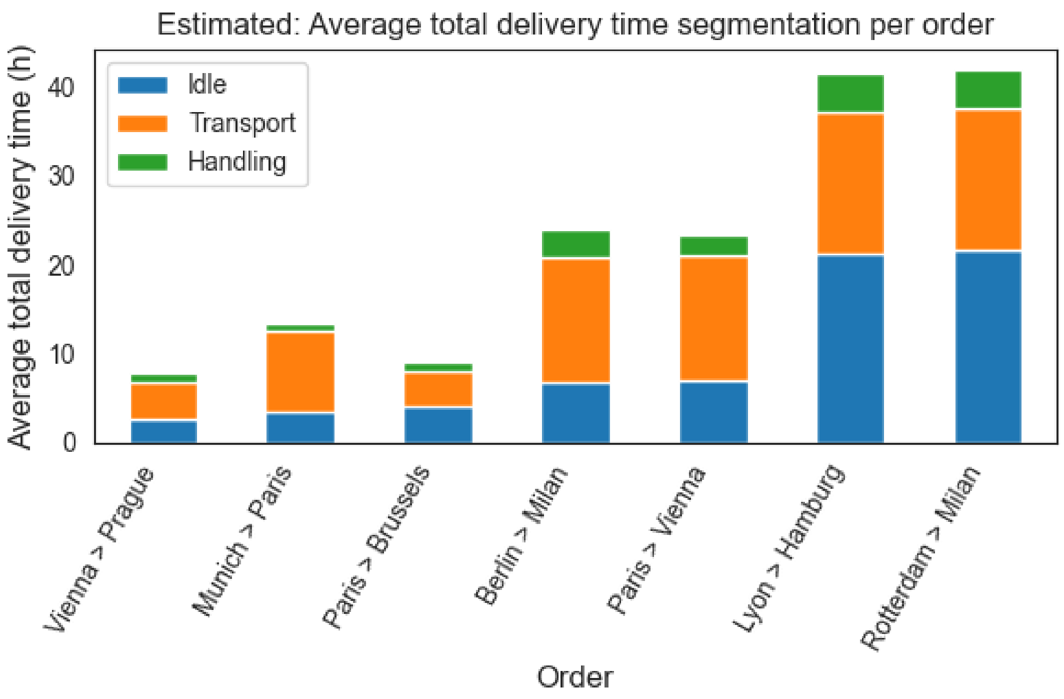


Figure 5.12: Estimated: average total delivery time segmentation per order (Network: Europe; Protocol: Volume; *orderfreq_mult*: 1).

This network is simulated for a range of *orderfreq_mult* values, which effectively tests how robust its performance is with a lower/higher volume of containers. The results are depicted in figure 5.13.

These show that the performance remains optimal between 0.5x and 1.5x the normal container volume. When the multiplier is even lower than 0.5, the performance decreases. At 0.1, the on-time fraction becomes 54.72[50.4,59.05]%.

This chart also gives the false impression that the performance gradually decreases when the multiplier becomes larger than 1.5. This is not the case, as these higher multipliers cause the containers to build up. They never reach a steady state, as indicated in figure 5.2. Thus, the on-time delivery fraction should actually be zero for multipliers higher than 1.5. The buildup just happens faster for larger volumes, hence why the fraction seems to gradually go towards zero.

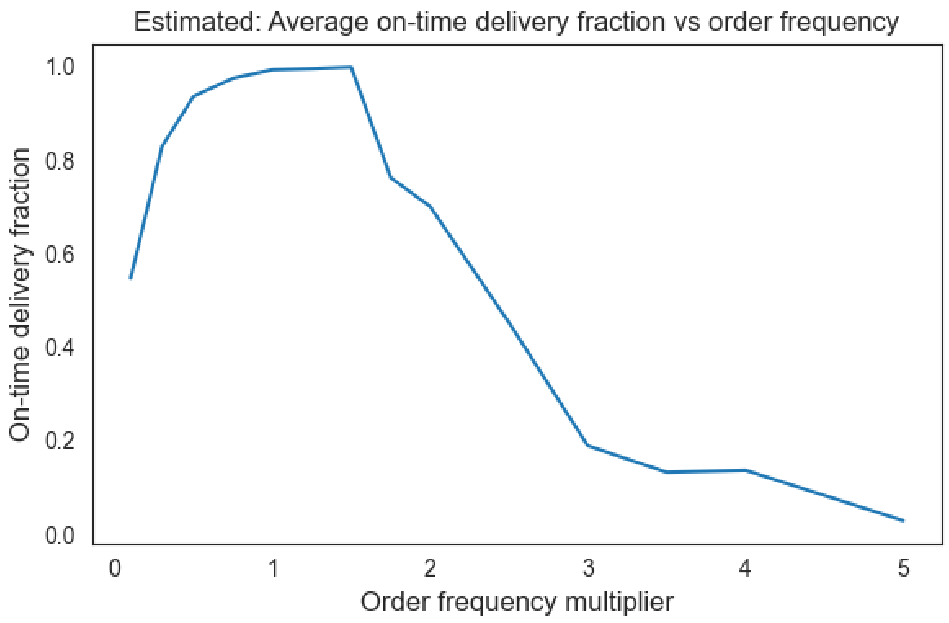


Figure 5.13: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult* (Network: Europe; Protocol: Volume).

This buildup happens because there are not enough trucks to transport all containers. The PI network successfully scales the number of truck trips up to a multiplier of 1.5. This is visible in figure 5.14, which shows that the total driving time grows at the same rate between a multiplier of 0 and 1.5. At 1.5 however, the graph reaches an inflection point. After this point, the total driving time does not increase fast enough to cope with the increased demand.

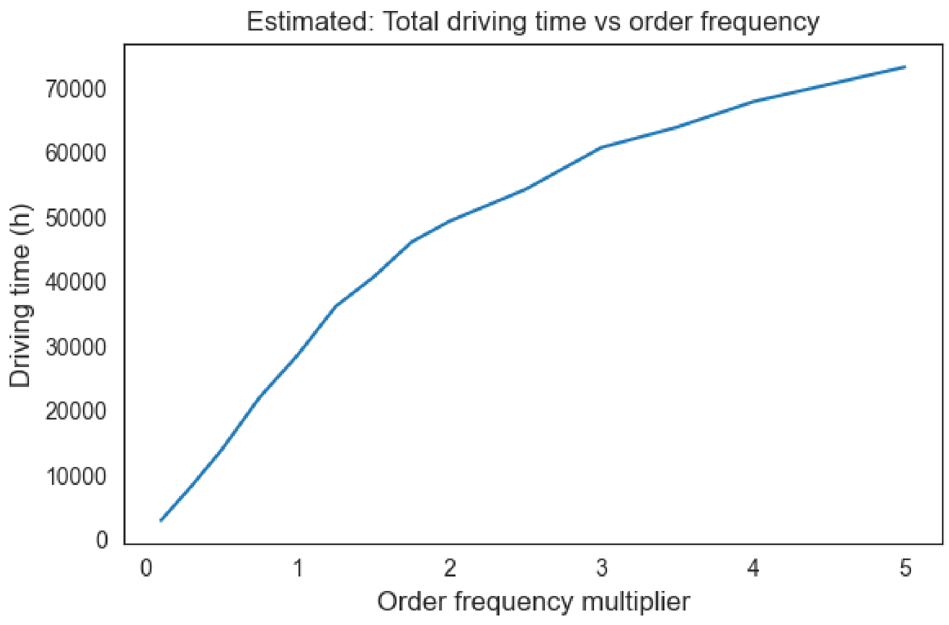


Figure 5.14: Estimated: Total driving time as a function of *orderfreq_mult*. (Network: Europe; Protocol: Volume).

The cause for the decrease in performance on the left side of the curve in figure 5.13 was located using these KPIs. Recall the correlation that was seen in figure 5.10 between the size of the confidence interval and the on-time delivery fraction. This correlation became even more apparent when visualizing the results for an *orderfreq_mult* of 0.3.

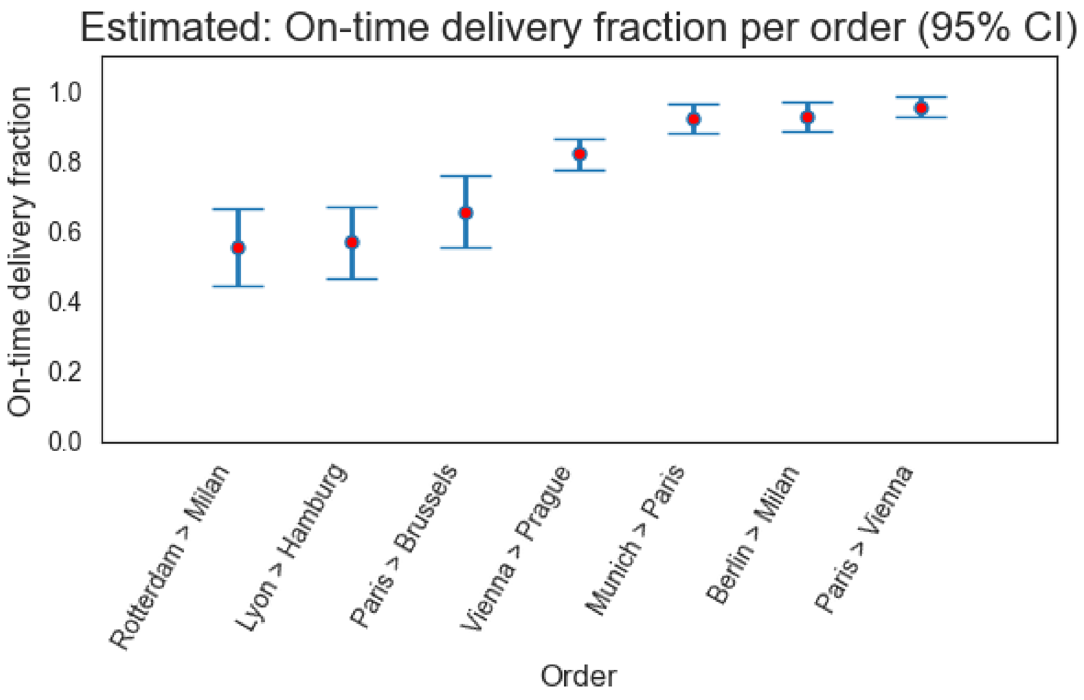


Figure 5.15: Estimated: On-time delivery fraction per order (Network: Europe; Protocol: Volume; *orderfreq_mult*: 0.3).

Apparently, orders that have a lower volume (and thus produce less samples, which leads to a larger confidence interval), arrive too late more often. This relation is confirmed by plotting the relation between the on-time fraction and the order *interval* parameter. The correlation gets stronger when the volume is lowered, as shown in Figures 5.16 and 5.17.

The explanation for this phenomenon is quite simple: when containers are generated at a slow rate, trucks do not get filled quickly enough. By the time enough containers have been generated to trigger the dispatch rule, some of them are already too late to arrive at their destination before the deadline. This happens more often for containers with a large interarrival time. Assigning a small value to *orderfreq_mult* effectively increases the interarrival time for all containers.

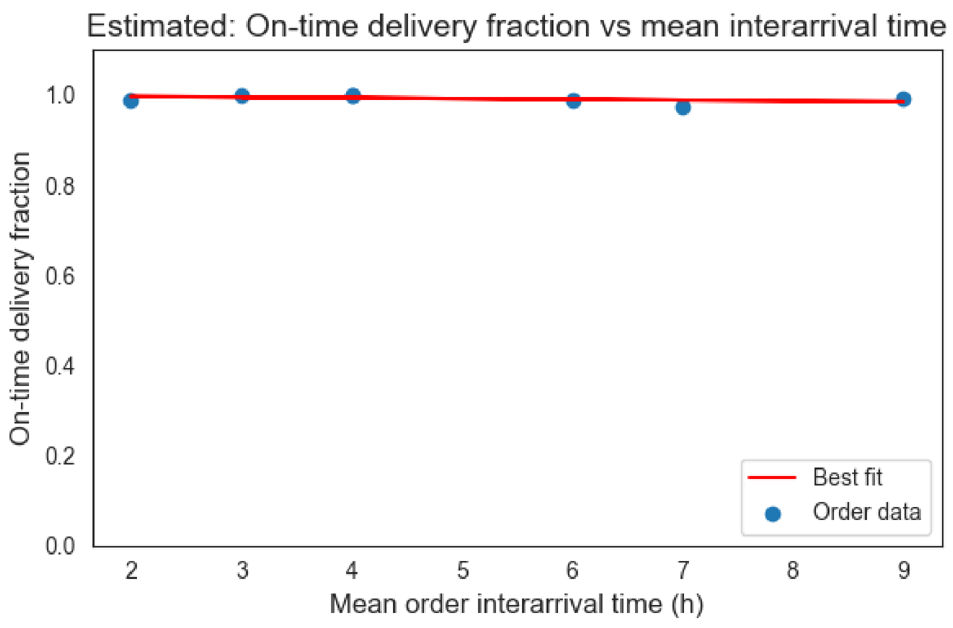


Figure 5.16: Estimated: On-time delivery fraction as a function of mean interarrival time. (Network: Europe; Protocol: Volume; *orderfreq_mult*: 1).

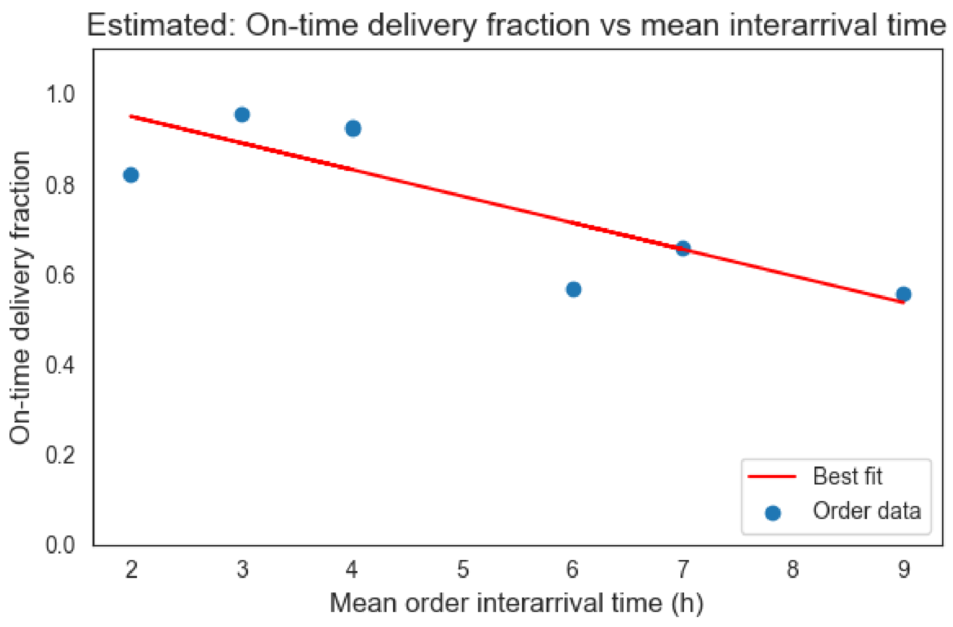


Figure 5.17: Estimated: On-time delivery fraction as a function of mean interarrival time. (Network: Europe; Protocol: Volume; *orderfreq_mult*: 0.3).

This analysis is repeated for the small-scale network in Belgium. All results follow a similar trend. However, there are two key differences.

Firstly, the average truck load fraction is a lot higher. In the small-scale network, the flow of containers is more balanced throughout the network. A lot of edges have flows in both directions, which gives returning trucks more opportunities to carry containers. This results in an average load fraction of 62.99%, compared to 56% in the larger network.

Secondly, figure 5.18 shows that the average handling time is almost as large as the average transport time. As expected, handling makes up a larger segment of the total delivery time than in the large-scale network (figure 5.12).

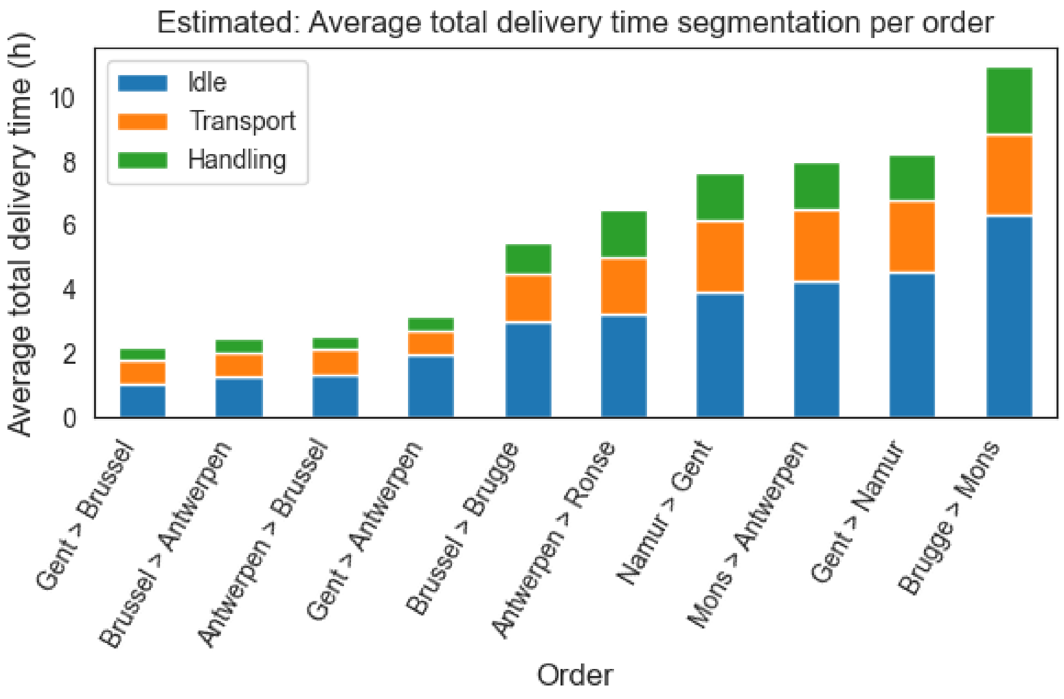


Figure 5.18: Estimated: average total delivery time segmentation per order (Network: Belgium; Protocol: Volume; *orderfreq_mult*: 1).

5.3.5 Rule 3: Return trucks to parent after patience runs out

The next rule is an edit of the first rule that attempts to increase the average truck load fraction. In contrast to the first rule, it now contains a *patience* parameter. When a truck leaves its parent node and arrives at its destination, it is no longer immediately returned. Instead, the truck driver waits up to *patience* hours for their truck to be filled. If the truck is not completely filled in this interval, the driver leaves with the currently loaded cargo.

5.3.6 Protocol 2: Patience

The second protocol first checks rule 3 and then looks at rule 2. Its influence on the KPIs is studied in each network for several *patience* values. The total driving time is normalized. Note that a patience of zero makes the protocol equivalent to the first protocol. The results are displayed in figure 5.19 and 5.20.

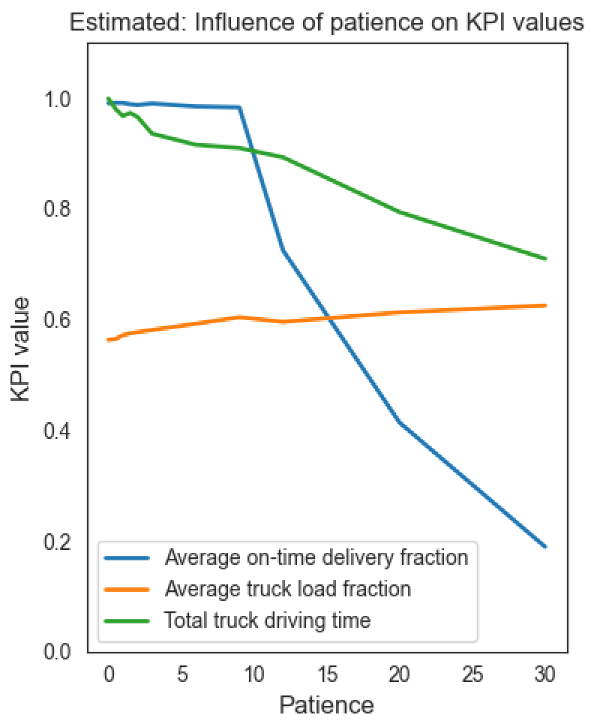


Figure 5.19: Influence of patience on KPI (Network: Europe; Protocol: Patience).

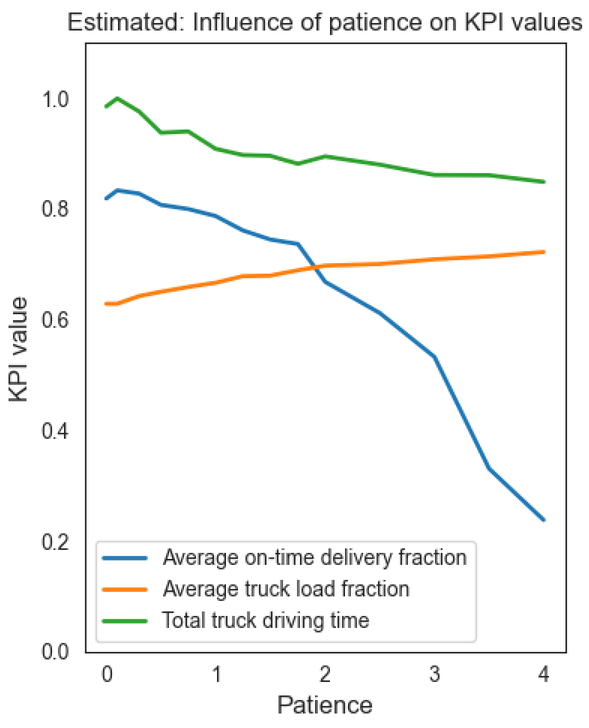


Figure 5.20: Influence of patience on KPI (Network: Belgium; Protocol: Patience).

In the Europe network, a patience value of 6 hours reduces the total driving time needed by approximately 8.4%. Moreover, it does this without significantly affecting the on-time delivery fraction, which only decreases from 99.13[98.84,99.43]% to 98.56[98.17,98.94]%. It is interesting to note that this driving time reduction is caused by an increase of only 5% in average load fraction.

These graphs show diminishing improvements in the load fraction. Both networks contain edges that only have container flow in one direction, which will never achieve a load fraction higher than 50%. This means there is an upper limit to the fill rate, which leads to the asymptotic growth depicted in these curves.

Furthermore, both figures show that the on-time delivery fraction reduces after a certain point. This happens because the returning trucks are losing too much time sitting idle, instead of coming back and providing transport capacity to their parent node. This drop-off in performance happens a lot earlier in the small-scale network, because the edge weights are much smaller. This is clearly caused by the fact that an hour of idle time has much more impact on a 50 minute round trip than on a 10 hour journey.

5.3.7 Rule 4: Dispatch urgent containers

It was discovered in section 5.3.4 that the volume-based protocols perform poorly when the order frequency is lowered. This happens because trucks get dispatched too late, because they are only allowed to leave when they are completely full.

Therefore, a rule is developed to counter this phenomenon. The urgency rule makes sure containers are sent out in time to reach their destination before their due date. It does so by calculating a dispatch deadline for the most urgent item of each heap. This dispatch deadline is calculated as follows:

$$dispatch\_deadline = due\_date - delivery\_time\_estimate$$

The protocol estimates the total delivery time that a container needs to arrive at its destination and subtracts it from their due date. This estimates the last possible time a container can leave and still arrive on time. The delivery time estimate is defined as:

$$delivery\_time\_estimate = \sum_{e \in P} [w_e + scale \cdot 2 \cdot truck\_storage\_capacity \cdot handling\_time]$$

Where P is the set of edges in the shortest path to that container's destination and w_e is the weight of edge e. At every edge on that path, the container needs to be loaded on a truck at the start and unloaded at the end. This processing time takes at least $truck\_storage\_capacity \cdot handling\_time$ hours.

This estimate only represents a lower bound on the delivery time, because it does not consider waiting on forklifts and other idle time spent at intermediary nodes. In reality, almost all containers will need more time to arrive at their destination. If this lower bound is used, the containers that are sent out by the urgency rule often arrive just after their deadline. Therefore, a *scale* parameter is added to adjust the estimate. In this dissertation, this parameter is set to 1.2, which was calibrated by comparison.

5.3.8 Protocol 3: Urgency

The third protocol contains rules 1, 2 and 4. It essentially first executes the volume-based protocol and then checks if the urgency rule is triggered. Figures 5.21 and 5.22 show the estimated distribution of all container's lateness values in the Europe network with low volume.

The first image is generated using the volume protocol. Its estimate of the lateness distribution has a right tail of containers that arrive after their due date. When the urgency rule is implemented, it makes sure these containers are sent out in time. This results in figure 5.22, where the long tail has moved back into the buckets of containers that have a negative lateness and thus arrived on time.

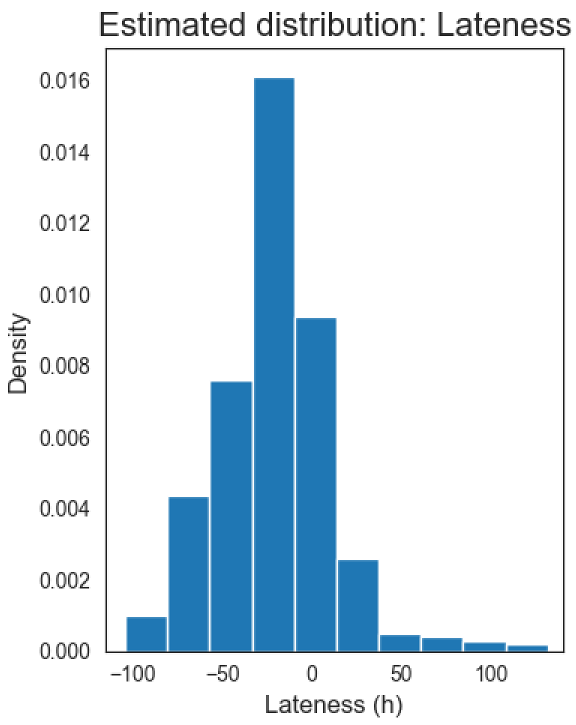


Figure 5.21: Estimated lateness distribution (Network: Europe; Protocol: Volume) (*orderfreq_mult*: 0.3).

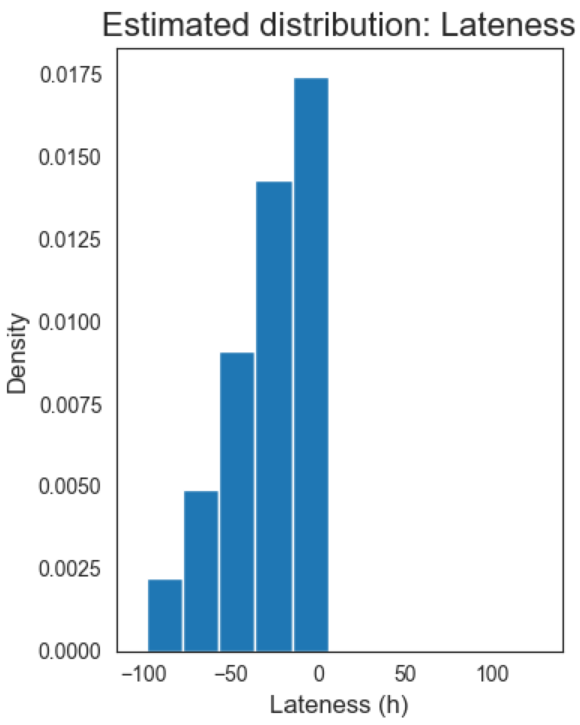


Figure 5.22: Estimated lateness distribution (Network: Europe; Protocol: Urgency) (*orderfreq_mult*: 0.3).

This urgency protocol is then simulated for multiple values of *orderfreq_mult* and the result is shown in figure 5.23. It clearly shows the impact of adding the urgency rule to the volume protocol. The drop in on-time delivery fraction at low order frequencies has been solved. These simulations now deliver all of their containers on time. This performance is to be expected, as the network can already achieve a perfect performance at an order frequency of 1.5. If it has enough capacity for this amount of containers, any amount smaller than that should also work.

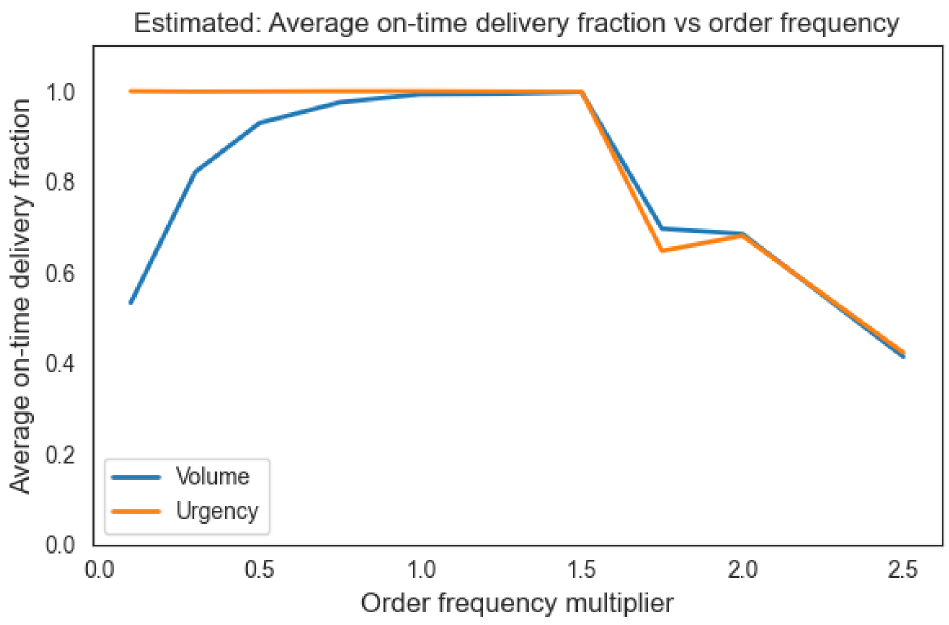


Figure 5.23: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult* (Network: Europe; Protocol: Volume and Urgency).

Since trucks are now being sent even if they can not be completely filled, the average truck load fraction decreases. Figure 5.24 visualizes this effect. As a consequence of this load fraction decrease, the total amount of trips increases. Figure 5.25 shows the growth in total driving time compared to a network that uses the volume protocol.

The results for the Belgium network are very similar and are therefore not presented.

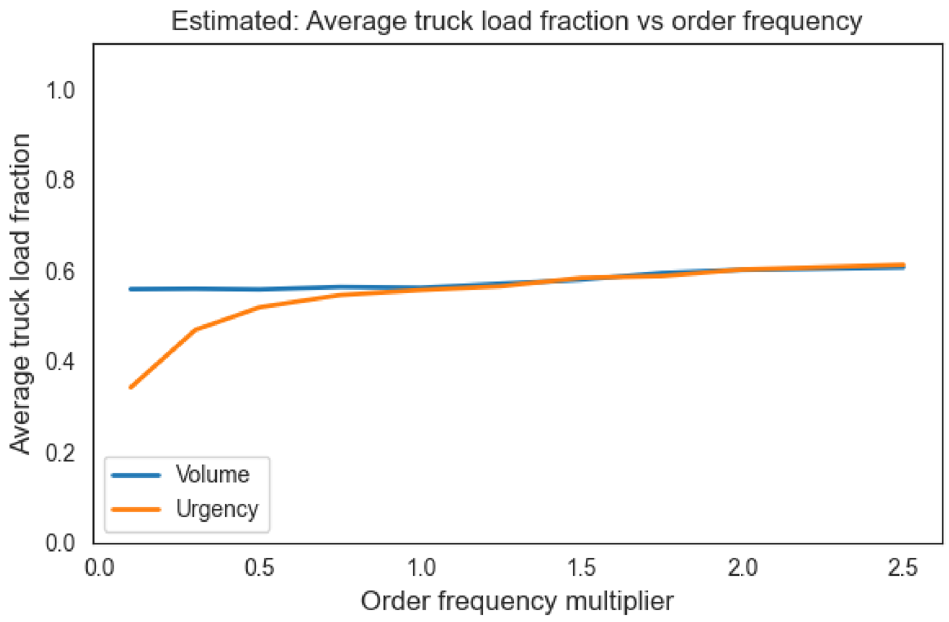


Figure 5.24: Estimated: Average truck load fraction as a function of *orderfreq_mult* (Network: Europe; Protocol: Volume and Urgency).

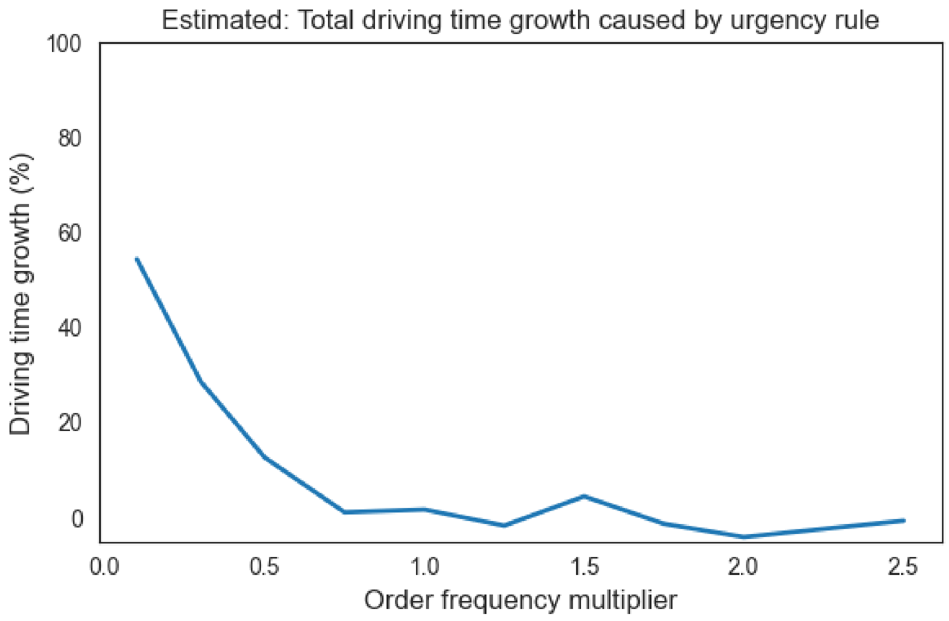


Figure 5.25: Estimated: Total driving time growth caused by urgency rule (Network: Europe).

5.4 Dynamic routing protocols

In this last section, dynamic routing protocols are researched. Even though the assumption of constant network topology and edge weights still holds, these protocols do change their routing tables during the simulation.

They achieve this in two ways. The first protocol type queries the State Graph to retrieve relevant information about neighboring nodes. It uses this information to make a more informed routing decision. The last protocol allows suboptimal paths to be taken if more containers can be transported that way.

5.4.1 Inventory structure

Since the routing is now dynamic, protocols can no longer group their inventory by next destination. The path of a node is always decided by looking at its target node. Therefore, these protocols partition their containers by their final destination.

This increases the complexity of the previously defined rules, which still make a decision based on which neighbor is considered. To circumvent this, each protocol iteration starts by allocating heaps to the neighbor that they should pass through.

For example, the volume rule always checks if there are enough containers that need to go in the same direction. Instead of looking at that neighbor's heap, it now checks the sum of the heaps allocated to that neighbor.

5.4.2 Increased information scope: Incoming trucks

One of the objectives of the framework, presented in section 3.2, is to enable the testing of protocols that utilize state information to improve their performance. Protocol 4 queries the *incoming_trucks* attribute of its neighbors to estimate the time a truck will spend waiting for a forklift. The *incoming_trucks* attribute of a node contains the arrival time of every truck that is currently headed for that node.

In theory, this information can be used to simulate the forklift queuing system at the target node. It should be noted that this information only provides a lower bound on the arrivals in the queuing system, as outgoing trucks are put in the same queue.

When a protocol iteration starts, it first estimates when a truck would arrive at the target node if it was dispatched now. Next, it uses this forklift queuing simulation to calculate a lower bound on the waiting time it would experience. This is done by adding the estimated arrival time as one of the arrivals in the simulation and checking when it would be serviced.

In practice however, this 'simulation in a simulation' approach is difficult to implement and drastically increases the time complexity of the PI simulation. Therefore, an algorithm is developed that allows the protocol to calculate a lower bound on the waiting time in a fraction of the runtime. Pseudocode for this algorithm is provided in Algorithm 1.

Algorithm 1 Forklift waiting time estimate

```
processing_time = handling_time $\cdot$ truck_storage_capacity
for neighbor $\in$ neighbors do
    estimated_arrival = now + processing_time + Graph[source][neighbor]['weight']
    delaying_trucks = [ ]
    for arrival $\in$ neighbor.incoming_trucks do
        if estimated_arrival - processing_time < arrival < estimated_arrival then
            delaying_trucks.append(arrival)
    if len(delaying_trucks) > forklift_capacity then
        first_arrival = min(delaying_trucks)
        min_unavailable = processing_time $\cdot$ (len(delaying_trucks) // forklift_capacity)
        min_delay = first_arrival + min_unavailable - estimated_arrival
        if min_delay > 0 then
            Graph[source][neighbor]['weight'] += min_delay
```

This algorithm simplifies the forklift queuing system in two ways: (1) It only considers arrivals in the interval [*estimated_arrival* − *processing_time*; *estimated_arrival*], which are stored in *delaying_trucks*. These trucks will each require a forklift and keep it occupied during a period of *processing_time*. Since the trucks arrived after *estimated_arrival* - *processing_time*, they will always release their forklift after *estimated_arrival*. Therefore, they will possible delay the servicing of the dispatched truck. (2) All trucks in this interval are assumed to arrive together. Their arrival is chosen as the earliest arrival in the *delaying_trucks* list, which is stored in *first_arrival*. Doing this will underestimate the delay caused by each truck that arrived later in the interval and will thus produce a lower bound. Since all trucks have the same arrival time, simulating their servicing becomes much simpler. The process is presented as a flowchart in figure 5.26.

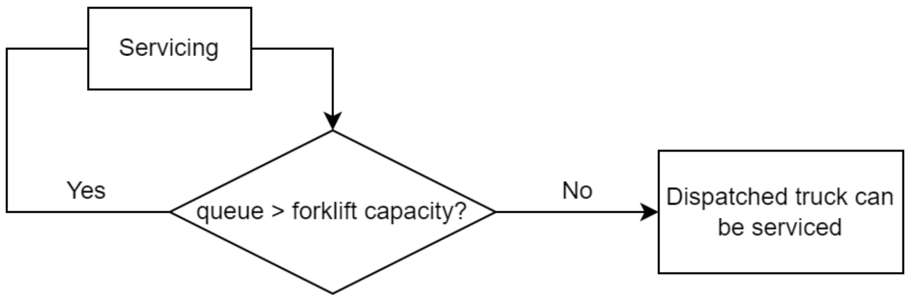


Figure 5.26: Servicing *delaying_trucks*

If the queue is larger than the number of available forklifts, each forklift will be unavailable during an interval of *processing_time* hours. In this interval, they each service one truck in the queue. This continues until the amount of trucks in the queue becomes smaller than the number of available forklifts. At this point, there will always be at least one forklift available to service the dispatched truck, which should be next in the queue.

The number of unavailability intervals is calculated by dividing the number of trucks in the queue by the amount of forklifts and taking the integer part. The sum of these intervals gives a lower bound on the time that the *delaying_trucks* block other trucks from being serviced. This is stored in *min_unavailable*. Adding this value to *first_arrival* gives the first possible time at which the truck after *delaying_trucks* can be serviced. If this time is larger than *estimated_arrival*, the dispatched truck will be delayed. If this is the case, the weight value of the edge to this neighbor is increased by *min_delay*.

5.4.3 Protocol 4: Information

The fourth protocol uses the same rules as the first, volume-based protocol. Before each iteration, it estimates the delays that trucks will experience and adjusts the edge weights.

This protocol is first simulated on the Europe & Belgium networks. It does not have any effect on their performance for two reasons: (1) the delays are often negligible compared to the distances between nodes (especially in the Europe network) and (2) there is a lack of alternative paths that are only slightly sub-optimal.

Therefore, a new testing network is used that should benefit from this protocol. The network is appropriately named 'Test' and consists of only 4 nodes. Its topology and flows using the 'Volume' protocol are shown in figure 5.27.

The network is engineered specifically to trigger dynamic routing: (1) It has a large *handling_time* of 0.3. (2) It contains an order from Gent to Hasselt, that can go through Brussels or Antwerp. The route through Brussels is only 0.1 hours shorter. (3) There is a large flow of containers from Hasselt to Brussels, which makes sure that Brussels has a lot of *incoming_trucks* at any time.

When this network is simulated using the 'Information' protocol, containers are indeed routed through Antwerp to avoid the congestion in Brussels. These flows are visualized in figure 5.28.

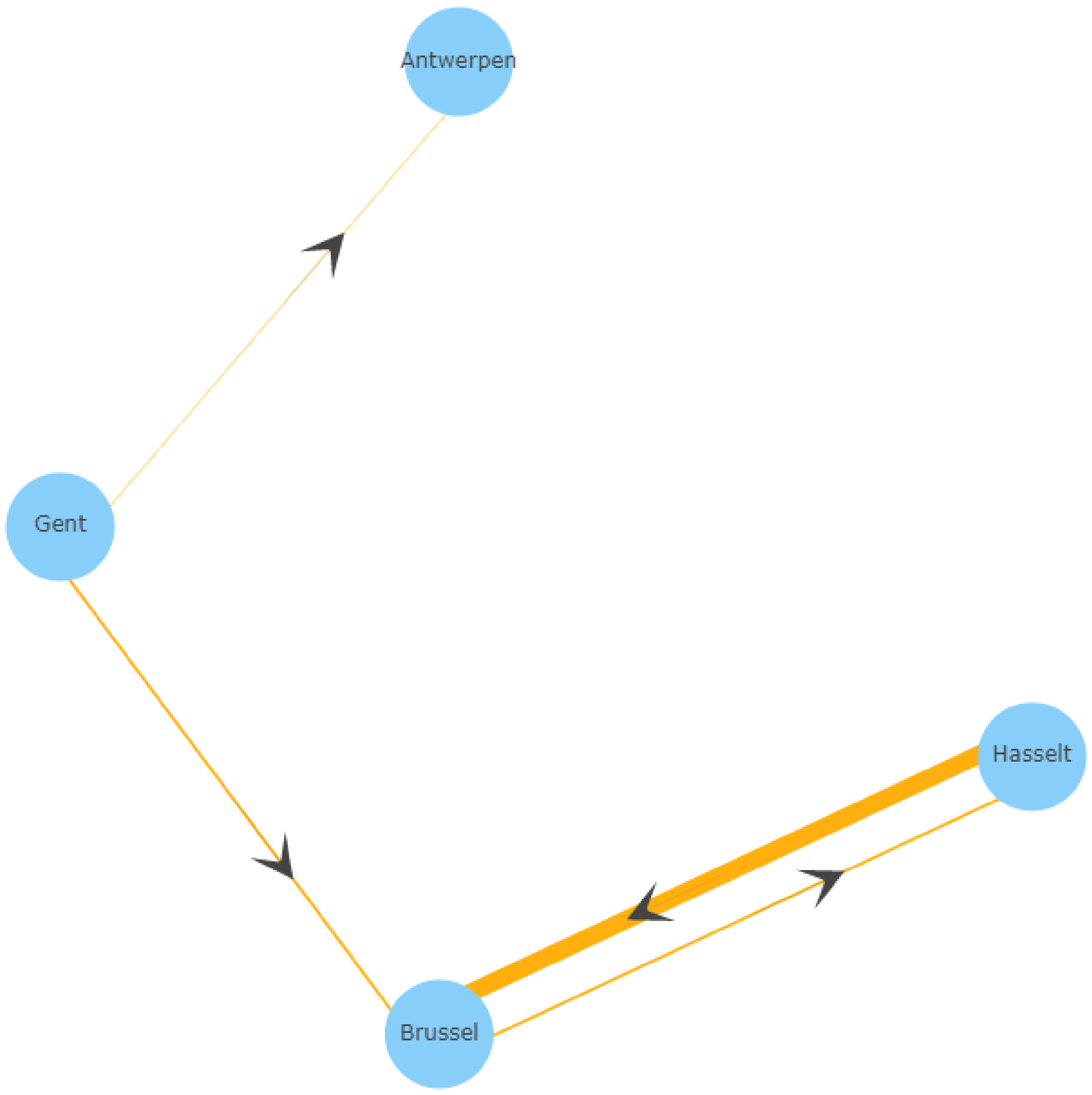


Figure 5.27: Test network
(Protocol: Volume; *orderfreq_mult*: 0.75).

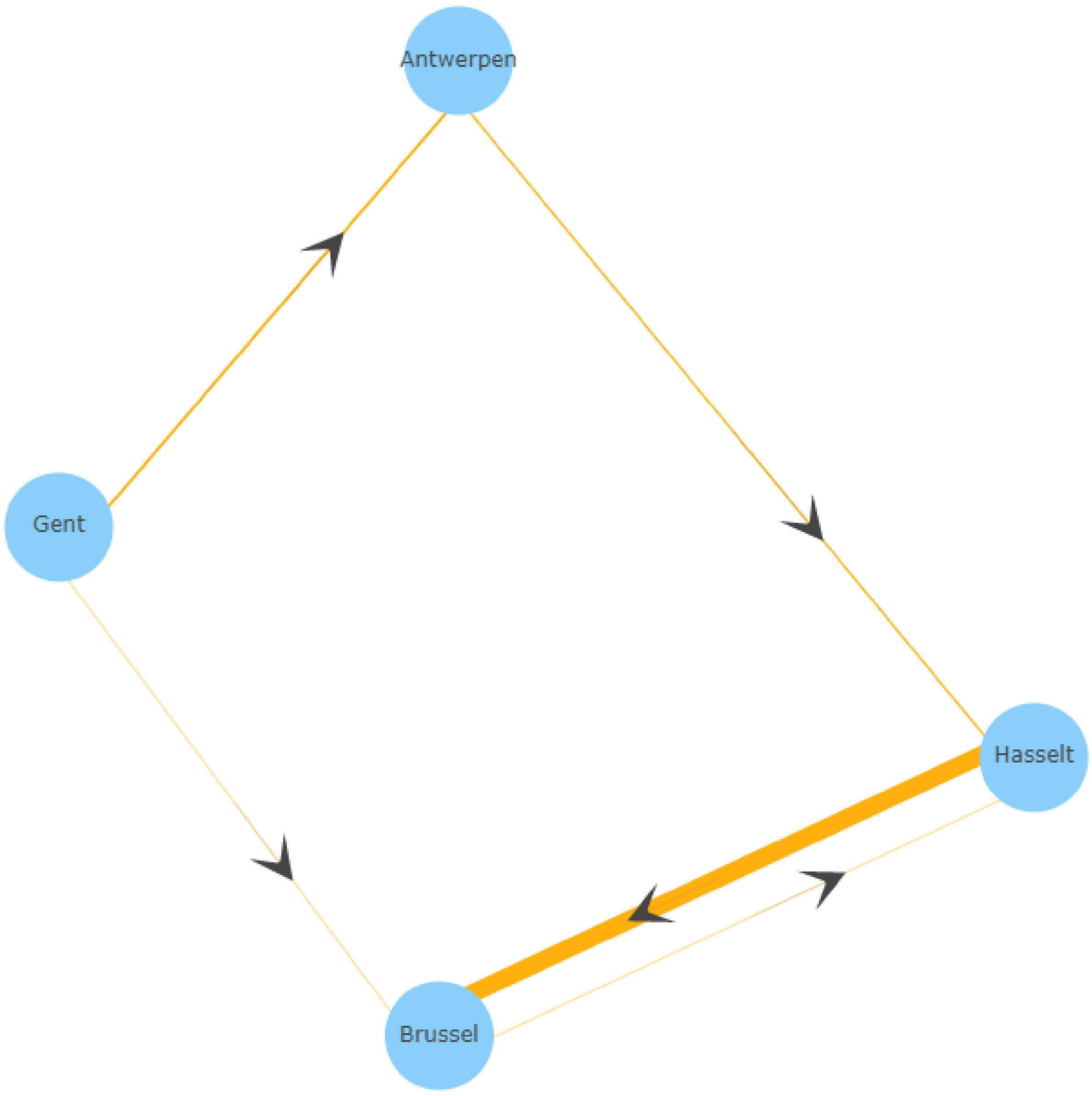


Figure 5.28: Test network
(Protocol: Information; *orderfreq_mult*: 0.75).

This protocol has no effect on the on-time delivery fraction of the network when the order frequency is low, as shown in figure 5.29. This happens because there are never a lot of trucks being serviced at once.

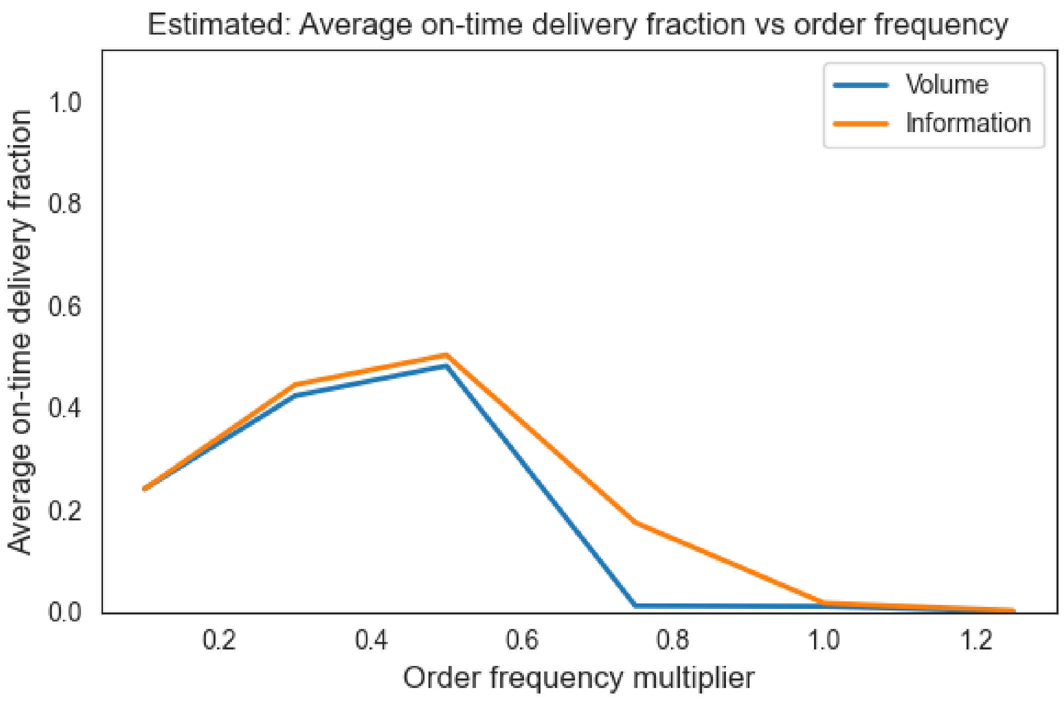


Figure 5.29: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult* (Network: Test; Protocol: Volume and Information).

Once the volume increases, the extra information significantly improves the on-time fraction. Figure 5.30 shows that the network with the 'Volume' protocol becomes saturated at an *orderfreq_mult* of 0.75. The new protocol successfully avoids congesting Brussels by offloading a part of the volume through Antwerp. It is still able to reach a steady state, as depicted in figure 5.31.

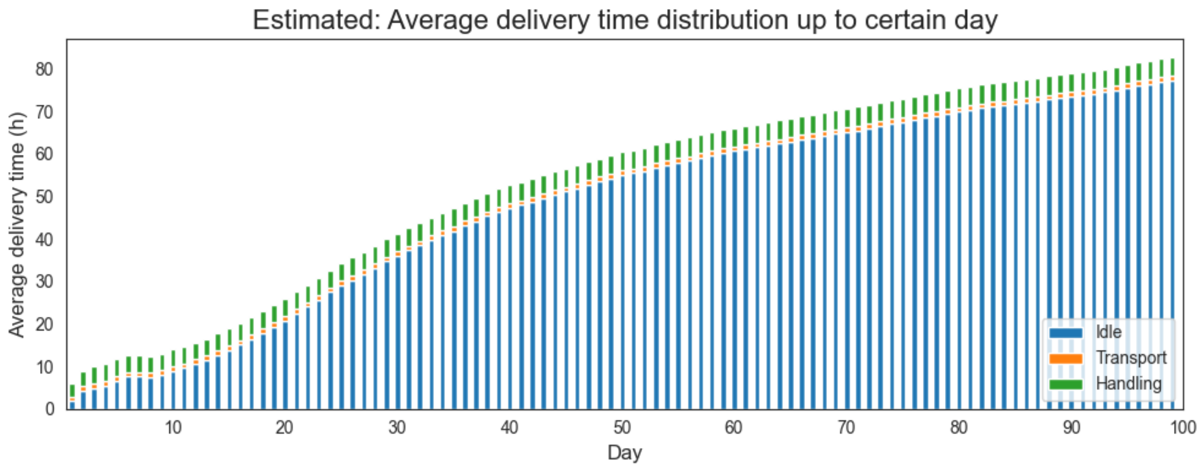


Figure 5.30: Estimated: Average delivery time evolution
(Network: Test; Protocol: Volume; *orderfreq_mult*: 0.75).

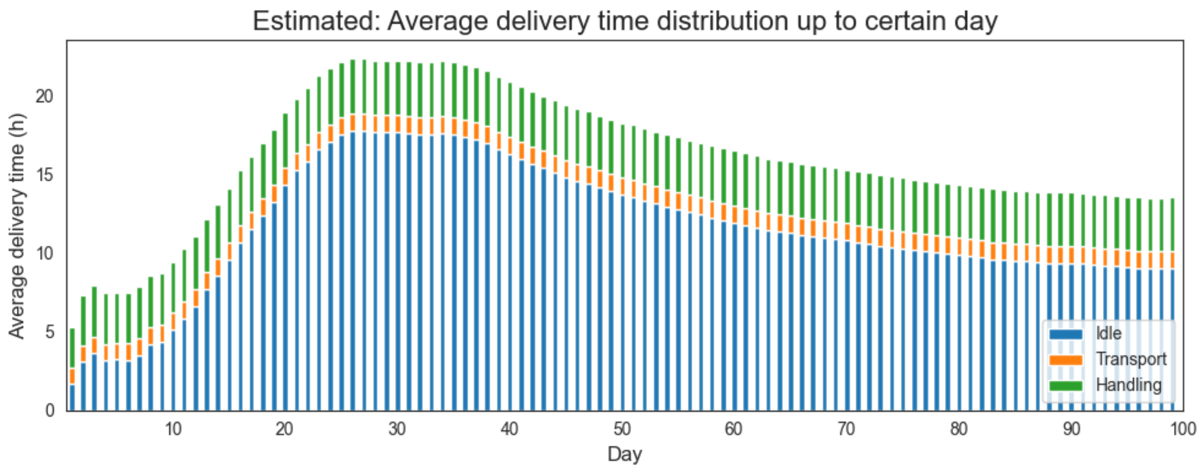


Figure 5.31: Estimated: Average delivery time evolution
(Network: Test; Protocol: Information; *orderfreq_mult*: 0.75).

5.4.4 Protocol 5: Information + Urgency

In the previous simulation, the left side of the on-time delivery fraction curve (figure 5.29) again shows a drop in performance. It has already been established in section 5.3.4 that this always happens for purely volume-based protocols.

Therefore, the increased information scope is used on top of the urgency protocol from section 5.3.8. This will show if the increased information scope still makes the network outperform when its low-volume problems are fixed. The results are displayed in figure 5.32.

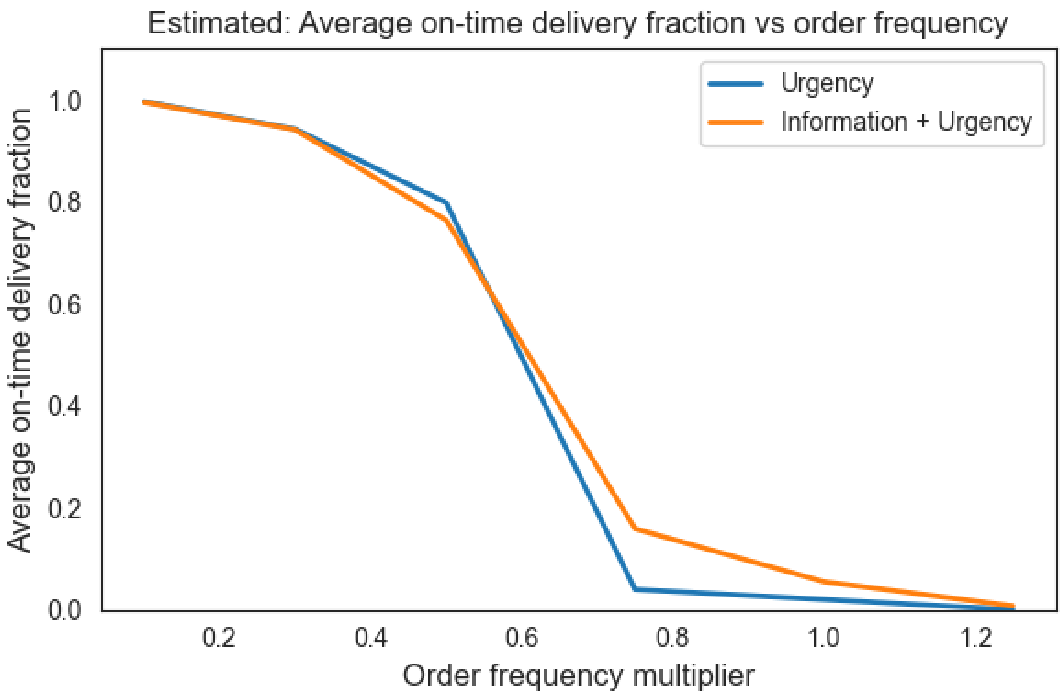


Figure 5.32: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult* (Network: Test; Protocol: Urgency and Information + Urgency).

As expected, the urgency rule improves the left side of the curve. The extra information also still does not have an effect on the performance in a low-volume network. Once more, it only outperforms when *orderfreq_mult* becomes 0.75. At this container volume, the alternative routing again saves the system from infinite congestion.

Sections 5.4.3 and 5.4.4 aimed to quantify the value of the information about incoming trucks. The analysis shows that it only has a large impact when the network is close to saturation.

Furthermore, it can be concluded that the value of this information depends heavily on the network topology and container volume. The value of a certain piece of information is thus best evaluated on a case-by-case basis. Fortunately, this framework allows users to easily edit the network and its parameters.

5.4.5 Suboptimal flow consolidation

Section 5.4.1 indicated that the node's containers are divided into heaps depending on their final destination. The protocol then allocates a heap to one of its neighbors if it is the next step in the journey of that heap's containers. These journeys are calculated by running Dijkstra's algorithm at the current node. This algorithm returns the length of the shortest path to each destination node, as well as its steps.

The next protocol deviates from this approach by introducing a suboptimality margin. For each heap, it first calculates the length of the optimal path to its target node. Then, it runs Dijkstra's algorithm at each neighbor. This enables the protocol to check how long the optimal path would be if it went through that neighbor. If the difference in length between that path and the optimal path is within the suboptimality margin, the heap is assigned to that neighbor.

This revised algorithm allows a heap to be allocated to multiple neighbors. This means that the containers in that heap are free to be routed through any of those nodes.

5.4.6 Protocol 6: Consolidation

The last protocol implements this revised heap allocation algorithm and is again tested on the 'Test' network. It first checks the default rules 1 and 2. Then, it increases the suboptimality margin to 0.1 hours. This enables containers from Ghent to Hasselt to be routed through Antwerp.

The performance of this protocol is again tested for different order frequency multipliers and compared to the simulations from section 5.4.4. It outperforms the previous protocols by a wide margin, without needing access to extra information. Because there is a separate order for containers from Ghent to Antwerp, the alternative route has become the standard. As it combines both flows, trucks towards Antwerp are often filled more quickly. This enables the alternative routing to happen at every value of *orderfreq_mult*, unlike the previous protocol. Please note that this alternative routing only happens because there is a separate order to Antwerp.

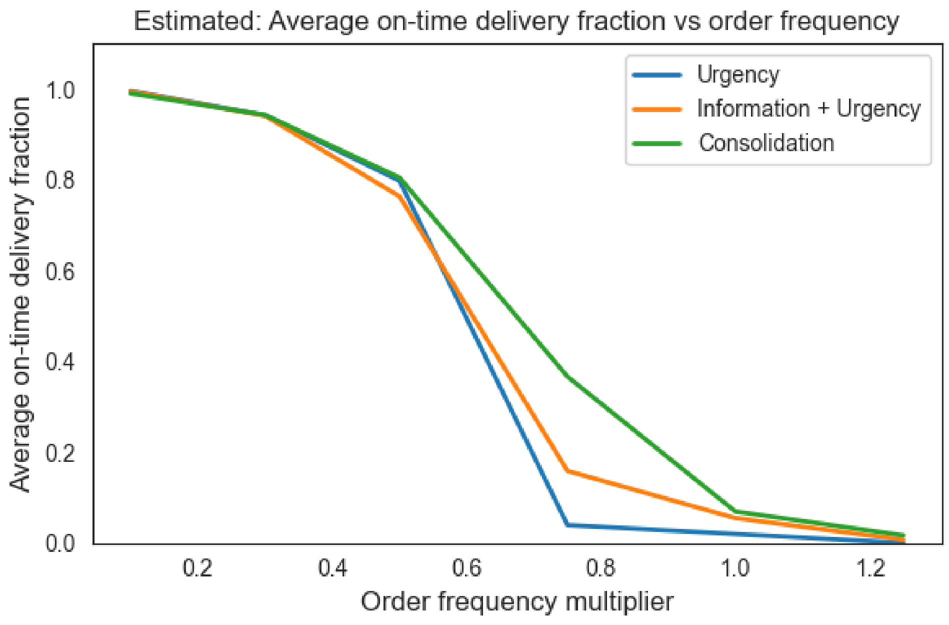


Figure 5.33: Estimated: Average on-time delivery fraction as a function of *orderfreq_mult* (Network: Test; Protocol: Urgency, Information + Urgency and Consolidation).

This chapter has given a demonstration of the capabilities of this simulation framework by using it to implement and compare several routing and consolidation protocols.

Chapter 6

Conclusion and future research

This last chapter presents the conclusions of this research and lists potential topics for further research using the proposed simulation framework.

6.1 Conclusion

The primary goal of this research was to develop a modular, open-source framework for simulating PI-enabled networks. The first half of this dissertation described the development process. It started by generalizing the elements of a PI network and models these as agents. This model was then simplified by making some assumptions, which were thoroughly evaluated.

The proposed PI simulation toolbox allows research into routing and consolidation protocols. The second half of this research demonstrated this by using it to implement and analyze several protocols. First, it described the evaluation methodology and available diagnostic tools. Afterwards, a generalized rule-based protocol was defined. Next, static routing algorithms that do not use any outside information were designed and compared. The complexity was then increased by allowing dynamic routing and state information querying. From this analysis, it can be concluded that the value of certain information depends heavily on the network topology and container volume. It is thus best evaluated on a case-by-case basis.

Fortunately, this framework allows users to easily edit the network and its parameters. Even though its functionality could still be extended by relaxing assumptions, it has proven to be versatile and detailed enough to enable PI protocol research.

6.2 Future research

This section provides suggestions for future research that can leverage the proposed simulation framework. Firstly, a description of the concepts that could be investigated by adapting certain assumptions is given. The second research avenue consists of thoroughly exploring the impact of PI network design on performance indicators. The final topic encourages researchers to use this framework in the analysis of increasingly complex PI protocols.

6.2.1 Assumptions

In section 3.6, all assumptions made in the framework were presented. This section contains ways to potentially improve the simulation toolbox by editing or fully relaxing some of these assumptions. It also highlights future research avenues these changes could lead to.

Assumption 3: The network topology is constant

In this framework, it is assumed that the network edges and their weights remain fixed during the simulation. This is not a realistic assumption. First of all, the travel time between two places is a random variable. Its distribution should have a long tail to account for large disturbances like accidents and traffic jams. Secondly, the shape of this distribution greatly depends on the time of day and even on the period of the year.

It would be interesting to test the robustness of PI networks to changes in topology. One could investigate this by modeling the driving times as stochastic variables. This would add an extra source of randomness to the simulation, other than the random arrival of new containers.

Assumption 4: All transport is unimodal

The current model does not support alternative transport means. Enabling efficient multimodal transport is however one of the main selling points of the PI. It should be a responsibility of the protocols to choose which vehicle to use. To choose, they need certain criteria that they can use to evaluate the different options. Currently, only time-based KPIs are considered, which does not motivate the protocol to make a calculated choice.

For example, [27] states that trains mostly represent an opportunity to reduce the costs and environmental footprint of logistics. Modeling the cost and emission of each transport mean should therefore be important. This allows protocols to try to optimize delivery time, while minimizing emissions and costs. A score could be assigned to each of the KPIs, which allows the protocols to calculate a score-based weight for each transport mean on each edge. This score could then be used during routing and the influence of the weights of the different KPIs is another parameter to potentially research.

Assumption 5: All containers are identical

It is assumed that each container has the same dimensions. In reality, there will be a set of standardized dimensions. This set could be added to the model. The order generator will then randomly choose a size according to a certain distribution. The shape of this distribution could be a first topic for future research.

Having multiple sizes also necessitates a new optimization algorithm for the packing of a truck. Optimally filling transportation means with containers should be the responsibility of the decision making unit. [9] describes a separate packing protocol that uses Mixed integer linear programming (MILP) to solve the bin-packing problem. The use of other approaches could be insightful to research.

Assumption 6: All containers are fully packed

The containerization of individual physical goods is not considered in this framework. Future research could expand the decision making unit to include a containerization protocol that decides when to fill a container, which goods to bundle and which container size to use.

Assumption 7: Containers are an infinite resource

This simulation does not contain any capabilities to manage empty containers. Nevertheless, it is an important and complex research area that has not yet been fully examined in a PI setting. The following three topics in this domain are definitely worth investigating:

- The first topic is the development of autonomous protocols for container fleet management. These are responsible for bringing empty containers to source nodes in unbalanced networks. The transport of containers between China and the U.S.A is a distinctive example of an unbalanced network.
- Secondly, there has been no research done on the amount of containers that would be necessary in a PI network. There is a trade-off to be made when deciding which amount to use. Too little containers will cause bottlenecks in transport, while too many will result in large inventory costs.
- Finally, business models for container fleet management have not been thoroughly tested. These include the maintenance and replacement of containers. [28].

Assumption 8: Truck drivers and depot workers are available 24/7

The current framework assumes workers are always available at the same capacity and that truck drivers have no driving time constraints. Relaxing this constraint leads to multiple future research topics.

Firstly, the robustness of PI networks against sudden reductions in workers (e.g. sickness, strikes, etc.) can be tested by simulating the available capacity as a stochastic variable. Secondly, extensions can be added to the decision making unit and container generators that take into account the different employee shifts. For example, rules could be added to the node protocols that make sure all truck drivers return home by the end of their shift: their parent node will prevent them from leaving if they won't be able to get back in time and neighboring nodes will send them back even if their cargo is empty. This can quantify the loss in efficiency from having human drivers or, in other words, the gain in efficiency from switching to autonomous trucks.

Assumption 9: Trucks are serviced in a FIFO queue

Every incoming/outgoing truck that needs to be serviced is put in the same FIFO queue by the nodes in this PI framework. This assumption can cause problems when lots of trucks need to be serviced at once. For example, if a lot of loaded trucks arrive, the node will have to unload all of them before it is able to send out a truck of its own.

The optimization of this queuing system is another potential research topic. The decision making unit could be extended to contain a servicing protocol that decides which truck will first be handled. PI nodes should have a mechanism to prioritise certain trucks (e.g. trucks filled with high priority items) over others. A potential approach could be to split the queues by service type (load/unload) and designate forklifts for each activity.

Assumption 11: Container handling time is fixed

It is assumed that during loading/unloading of a truck, each container takes the same amount of time. This causes the simulation to underestimate the handling time of partially empty trucks and overestimate it for filled trucks.

This assumption can be revised to reduce this bias. For example, the handling time could be split into two parts. The first part is the time needed to drive a forklift to/from a truck and open/close it, which does not depend on the contents of the truck. The second part is the sum of all handling times of the containers.

[8] states that standardized containers should be able to interlock with each other. A forklift can then handle multiple of them at once. Therefore, the marginal difference in handling time will decrease when handling a larger number of containers. Figure 6.1 illustrates the difference between the current and revised assumption using arbitrary data.

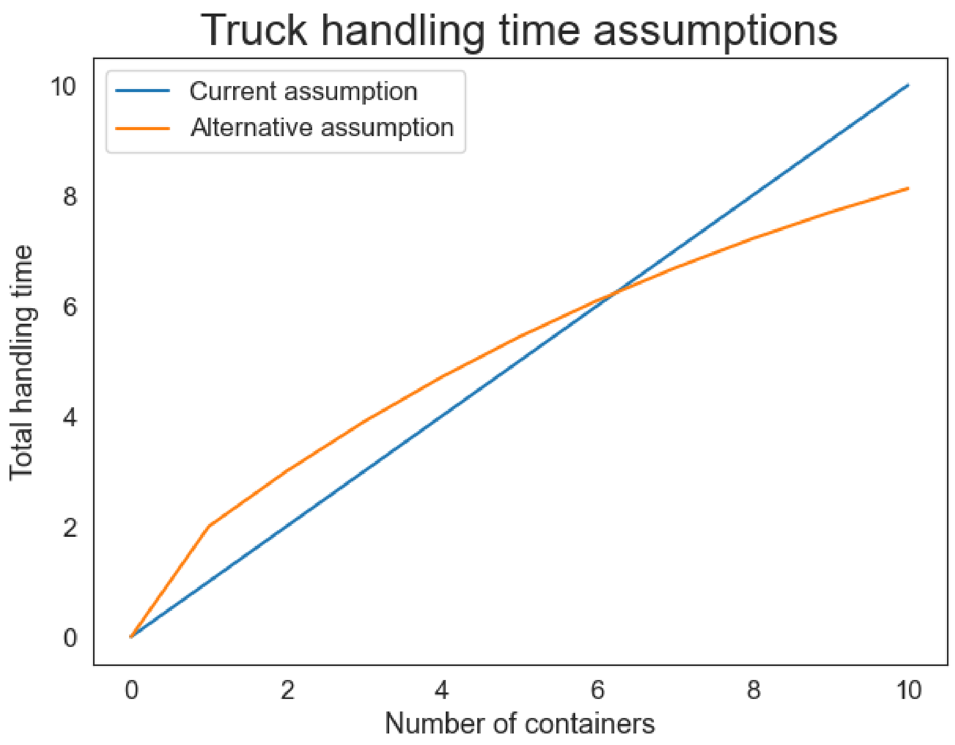


Figure 6.1: Revised truck handling time assumption.

6.2.2 PI network design

The second category of future research topics covers the design of PI networks and its effect on the KPIs. The placements and sizes of hubs and the connections between them have a large impact on the performance of the network. These set the boundaries within which protocols can route containers to their destination. It could be interesting to research a formalized version of this problem: *Given a set of overlapping supply chains with historical container flows, what is the optimal way to combine these in a PI setting?* Figure 6.2 illustrates this problem.

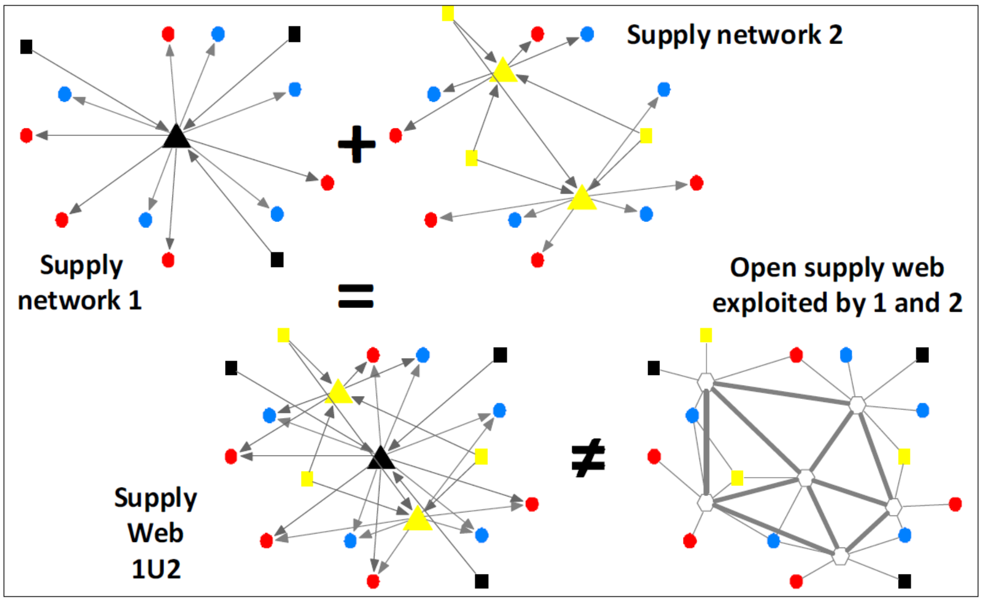


Figure 6.2: Comparing the combination of independently optimized supply networks with a optimized PI open hub network. [29]

Furthermore, the influence of demand unbalance on PI networks is currently not well understood. Networks where only one side is producing goods will probably not benefit as much from the PI, as most trucks will continue returning empty. It could be beneficial to formulate a parameter that quantifies how balanced the demand is. The influence of this parameter can then be studied by using the proposed simulation framework.

6.2.3 PI protocol research

The final recommended subject is a direct follow-up of the analysis in the second half of this dissertation. The literature review revealed that there is no existing research on the effect of using protocols with varying degrees of trust and collaboration. The simulation framework could be used to investigate the following two topics.

- There are still a lot of approaches left to explore with rule-based protocols that only manage a single node. For example, a protocol can be developed that queries not only the list of future truck arrivals, but also the future arrivals of newly generated containers. This information can be used to obtain a more detailed estimate of the idle time at neighboring hubs. The influence of the *lookahead* value could also be examined.
- This dissertation did not yet implement any large-scale optimization protocols. These control multiple nodes at once, or even the entire network. Doing so, they could achieve a higher level of collaboration and even plan departures in advance.

Bibliography

[1] T. Ambra, *The physical internet: A next-gen vision on logistics*, 2021. [Online]. Available: https://www.imec-int.com/en/articles/physical-internet-next-gen-vision-logistics.

[2] E. Ballot, S. Barbarino, B. van Bree, *et al.*, *Roadmap to the physical internet, executive version*, 2020. [Online]. Available: https://www.etp-logistics.eu/wp-content/uploads/2020/11/Roadmap-to-Physical-Intenet-Executive-Version_Final.pdf.

[3] B. Montreuil, *Physical internet manifesto: Globally transforming the way physical objects are handled, moved, stored, realized, supplied, designed and used*, 2009-2012. [Online]. Available: https://web.archive.org/web/20130120074150/http://physicalinternetinitiative.org/Physical%5C%20Internet%5C%20Manifesto_ENG_Version%5C%201.11.1%5C%202012-11-28.pdf.

[4] P. Markillie, "The physical internet: A survey of logistics.," *London: Economist Newspaper*, 2006.

[5] B. Montreuil, R. D. Meller, and E. Ballot, "Towards a physical internet: The impact on logistics facilities and material handling systems design and innovation," 2010.

[6] B. Montreuil, "Toward a physical internet: Meeting the global logistics sustainability grand challenge," *Logistics Research*, vol. 3, no. 2, pp. 71–87, 2011.

[7] J. El Ouadi, N. Malhene, S. Benhadou, and H. Medromi, "Shared public transport within a physical internet framework: Reviews, conceptualization and expected challenges under covid-19 pandemic," *IATSS Research*, vol. 45, no. 4, pp. 417–439, 2021, ISSN: 0386-1112. DOI: https://doi.org/10.1016/j.iatssr.2021.03.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0386111221000108.

[8] B. Montreuil, R. D. Meller, and E. Ballot, "Physical internet foundations," *IFAC Proceedings Volumes*, vol. 45, no. 6, pp. 26–30, 2012, 14th IFAC Symposium on Information Control Problems in Manufacturing, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120523-3-RO-2023.00444. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016331214.

[9] R. Sarraj, E. Ballot, S. Pan, D. Hakimi, and B. Montreuil, "Interconnected logistic networks and protocols: Simulation-based efficiency assessment," *International Journal of Production Research*, vol. 52, no. 11, pp. 3185–3208, 2014.

[10] B. Montreuil, E. Ballot, and F. Fontane, "An open logistics interconnection model for the physical internet," *IFAC Proceedings Volumes*, vol. 45, no. 6, pp. 327–332, 2012.

[11] S. Kaup, A. Ludwig, and B. Franczyk, *Framework artifact for the road-based physical internet based on internet protocols*, Jun. 2021.

[12] M. A. Majid, M. Fakhreldin, and K. Z. Zuhairi, "Comparing discrete event and agent based simulation in modelling human behaviour at airport check-in counter," in *International Conference on Human-Computer Interaction*, Springer, 2016, pp. 510–522.

[13] E. Ballot, B. Montreuil, R. Glardon, *et al.*, "Simulation de l'internet physique: Contribution à la mesure des enjeux et à sa définition," *PREDIT report to the 'Ministère de l'Écologie, de l'énergie, du Développement durable et de la Mer en charge des Technologies vertes et des Négociations sur le climat'of France*, p. 206, 2012.

[14] S. Gontara, A. Boufaied, and O. Korbaa, "Routing the pi-containers in the physical internet using the pi-bgp protocol," Oct. 2018, pp. 1–8. DOI: 10.1109/AICCSA.2018.8612885.

[15] S. Pan, E. Ballot, G. Q. Huang, and B. Montreuil, *Physical internet and interconnected logistics services: Research and applications*, 2017.

[16] M. Huschebeck, *Modulushca final report*, 2016. [Online]. Available: https://cordis.europa.eu/docs/results/314/314468/final1-modulushca-finalreport-sectiona.pdf.

[17] P. Kostov, *Iconet - innovation and networks executive agency*, 2019. [Online]. Available: https://ec.europa.eu/inea/en/horizon-2020/projects/h2020-transport/logistics/iconet.

[18] ICONET Consortium, *Living lab 1: Ports as the future engines of the physical internet and iot supply chains*, 2021. [Online]. Available: https://www.iconetproject.eu/wp-content/uploads/2021/05/ICONET-Living-Lab-1-Ports-as-the-future-engines-of-the-Physical-Internet-FINAL.pdf.

[19] ICONET Consortium, *Living lab 2: The potential for high volume transport corridors enabled by the physical internet*, 2021. [Online]. Available: https://www.iconetproject.eu/wp-content/uploads/2021/04/FINAL-VERSION-Living-Lab-2-Transforming-Ten-T-high-volume-freight-routes-into-IoT-enabled-corridors.pdf.

[20] ICONET Consortium, *Living lab 3: The role of iot in reducing costs and stock outs in fulfilment of online retail and home delivery*, 2021. [Online]. Available: http://www.iconetproject.eu/wp-content/uploads/2021/04/ICONET-Living-Lab-3-Home-delivery.pdf.

[21] ICONET Consortium, *Living lab 4: Physical internet as enabler towards networked warehousing as a service (waas)*, 2021. [Online]. Available: http://www.iconetproject.eu/wp-content/uploads/2021/04/ICONET-Living-Lab-4-Warehousing-as-a-Service.pdf.

[22] ICONET Consortium, *Iconet project website*. [Online]. Available: https://www.iconetproject.eu/.

[23] I. Danon and G. Poels, *Towards a physical internet ontology - exploring the boundaries of the internet metaphor*, 2020. [Online]. Available: http://lib.ugent.be/catalog/rug01:002837902.

[24] T. M. Pinho, J. P. Coelho, and J. Boaventura-Cunha, "Forest-based supply chain modelling using the simpy simulation framework," *IFAC-PapersOnLine*, vol. 49, no. 2, pp. 90–95, 2016, 7th IFAC Conference on Management and Control of Production and Logistics MCPL 2016, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.03.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896316300167.

[25] P. S. Goodman. "The real reason america doesn't have enough truck drivers." (), [Online]. Available: https://www.nytimes.com/2022/02/09/business/truck-driver-shortage.html. (accessed: 26.05.2022).

[26] N. Gazzard and B. Montreuil, "A functional design for physical internet modular handling containers," in *Proceedings of 2nd International Physical Internet Conference, Paris, France*, 2015, pp. 06–08.

[27] E. Ballot, B. Montreuil, and C. Thivierge, “Functional design of physical internet facilities: A road-rail hub,” 2012.

[28] P. Furtado, R. Fakhfakh, J.-M. Frayret, and P. Biard, “Simulation of a physical internet—based transportation network,” in *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, IEEE, 2013, pp. 1–8.

[29] E. Ballot, O. Gobet, and B. Montreuil, “Physical internet enabled open hub network design for distributed networked operations,” in *Service orientation in holonic and multi-agent manufacturing control*, Springer, 2012, pp. 279–292.

Appendix A

ETP-ALICE PI Roadmap

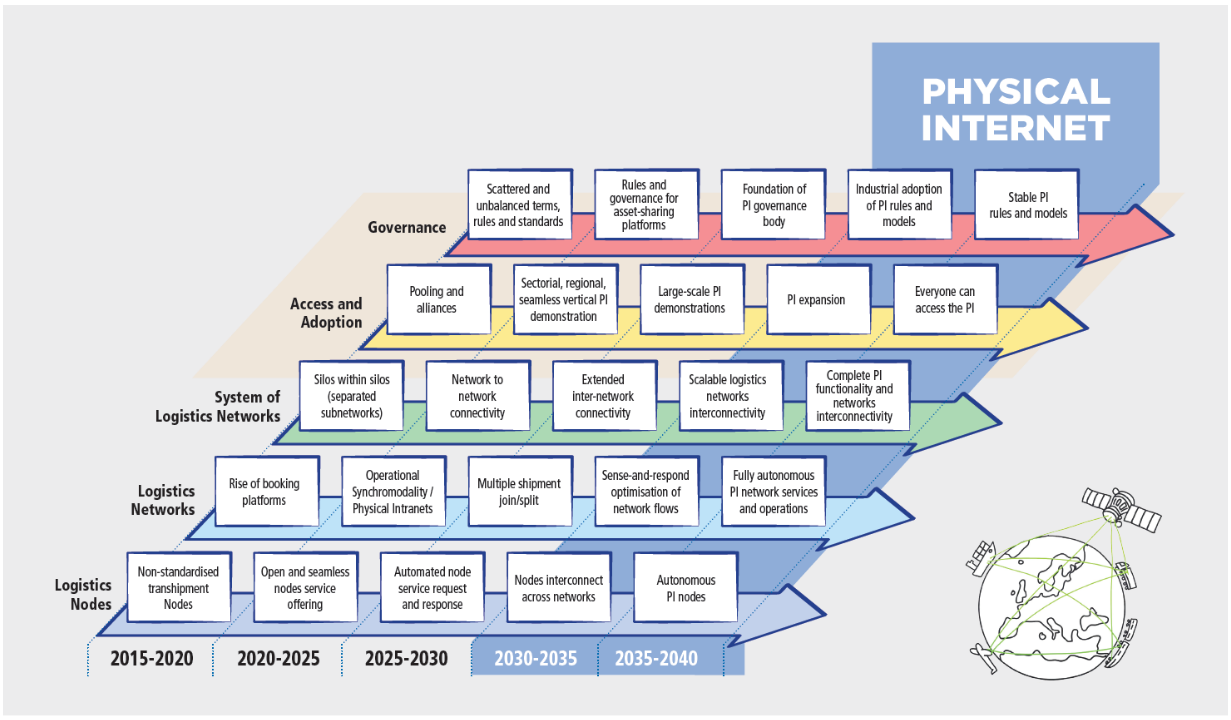

PHYSICAL INTERNET
Governance
Scattered and unbalanced terms, rules and standards
Rules and governance for asset-sharing platforms
Foundation of PI governance body
Industrial adoption of PI rules and models
Stable PI rules and models
Access and Adoption
Pooling and alliances
Sectorial, regional, seamless vertical PI demonstration
Large-scale PI demonstrations
PI expansion
Everyone can access the PI
System of Logistics Networks
Silos within silos (separated subnetworks)
Network to network connectivity
Extended inter-network connectivity
Scalable logistics networks interconnectivity
Complete PI functionality and networks interconnectivity
Logistics Networks
Rise of booking platforms
Operational Synchromodality / Physical Intranets
Multiple shipment join/split
Sense-and-respond optimisation of network flows
Fully autonomous PI network services and operations
Logistics Nodes
Non-standardised transhipment Nodes
Open and seamless nodes service offering
Automated node service request and response
Nodes interconnect across networks
Autonomous PI nodes
2015-2020
2020-2025
2025-2030
2030-2035
2035-2040

Figure A.1: The Physical Internet Roadmap. [2]

Appendix B

Input files

"Talk is cheap.
Show me the code."

Linus Torvalds

B.1 Network: Belgium

Table B.1: Network: Belgium, **parameters.csv**.

parameter	**value**
sim_days	120
truck_storage_capacity	3
handling_time	0.1
orderfreq_mult	1
lookahead	0
protocol_interval	0.5

Table B.2: Network: Belgium, **nodes.csv**.

name	**lat**	**lon**	**numforklifts**	**numtrucks**
Brugge	51.21125	3.216198	1	3
Gent	51.05183	3.751534	2	3
Antwerpen	51.22343	4.407419	2	3
Brussel	50.85258	4.364644	2	3
Kortrijk	50.82966	3.244715	1	3
Ronse	50.74854	3.5921	1	3
Charleroi	50.40996	4.4444	2	3
Mons	50.45678	3.947083	1	3
Namur	50.46972	4.860394	1	3

Table B.3: Network: Belgium, **edges.csv**.

source	**target**	**weight**
Brugge	Gent	0.75
Gent	Antwerpen	0.75
Gent	Kortrijk	0.5
Kortrijk	Ronse	0.5
Antwerpen	Brussel	0.75
Gent	Brussel	0.75
Brussel	Charleroi	1
Mons	Charleroi	0.5
Charleroi	Namur	0.5
Ronse	Mons	0.75

Table B.4: Network: Belgium, **orders.csv**.

source	**target**	**interval**
Antwerpen	Ronse	2
Brugge	Mons	2
Brussel	Brugge	2
Antwerpen	Brussel	2
Gent	Antwerpen	2
Gent	Brussel	2
Brussel	Antwerpen	2
Mons	Antwerpen	2
Namur	Gent	2
Gent	Namur	2

B.2 Network: Europe

Table B.5: Network: Europe, **parameters.csv**.

parameter	**value**
sim_days	120
truck_storage_capacity	3
handling_time	0.2
orderfreq_mult	1
lookahead	0
protocol_interval	1

Table B.6: Network: Europe, **nodes.csv**.

name	**lat**	**lon**	**numforklifts**	**numtrucks**
Paris	48.91457	2.34398	1	5
Brussels	50.9122	4.290115	1	5
Rotterdam	51.95838	4.478033	1	5
Berlin	52.605	13.32751	1	5
Hamburg	53.63413	9.855644	1	5
Prague	50.15851	14.42596	1	5
Milan	45.46271	9.181224	1	5
Vienna	48.27874	16.27098	1	5
Lyon	45.87799	4.793065	1	5
Munich	48.25042	11.65947	1	5

Table B.7: Network: Europe, **edges.csv**.

source	**target**	**weight**
Paris	Lyon	5
Paris	Brussels	4
Brussels	Rotterdam	2
Rotterdam	Hamburg	5
Hamburg	Berlin	3
Berlin	Rotterdam	7
Brussels	Berlin	8
Berlin	Prague	4
Prague	Vienna	4
Vienna	Milan	9
Lyon	Milan	5
Milan	Munich	6
Munich	Vienna	5
Munich	Prague	4
Paris	Munich	9

Table B.8: Network: Europe, **orders.csv**.

source	**target**	**interval**
Vienna	Prague	2
Rotterdam	Milan	9
Paris	Vienna	3
Paris	Brussels	7
Lyon	Hamburg	6
Munich	Paris	4
Berlin	Milan	4

B.3 Network: Test

Table B.9: Network: Test, **parameters.csv**.

parameter	**value**
sim_days	100
truck_storage_capacity	5
handling_time	0.3
orderfreq_mult	1
lookahead	0
protocol_interval	0.5

Table B.10: Network: Test, **nodes.csv**.

name	**lat**	**lon**	**numforklifts**	**numtrucks**
Gent	51.05182503	3.751534293	1	2
Antwerpen	51.22343357	4.407418567	1	2
Brussel	50.85258138	4.364643506	1	2
Hasselt	50.94315742	5.332987715	1	2

Table B.11: Network: Test, **edges.csv**.

source	**target**	**weight**
Gent	Antwerpen	0.75
Gent	Brussel	0.75
Antwerpen	Hasselt	1.1
Brussel	Hasselt	1

Table B.12: Network: Test, **orders.csv**

source	**target**	**interval**
Gent	Antwerpen	7
Hasselt	Brussel	0.3
Gent	Hasselt	2

Appendix C

Output file samples

Table C.1: Sample output file **transports.csv**.

truckname	**startnode**	**endnode**	**starttime**	**endtime**	**load**	**capacity**
Hasselt_0	Hasselt	Brussel	8	9	5	5
Gent_0	Gent	Antwerpen	9.5	10.25	5	5
Hasselt_1	Hasselt	Brussel	10	11	5	5
Hasselt_0	Brussel	Hasselt	10	11	0	5
Gent_0	Antwerpen	Gent	11.5	12.25	0	5
Hasselt_1	Brussel	Hasselt	12	13	0	5
Hasselt_2	Hasselt	Brussel	13	14	5	5
Hasselt_2	Brussel	Hasselt	15	16	0	5
Hasselt_3	Hasselt	Brussel	19	20	5	5
Hasselt_3	Brussel	Hasselt	21	22	0	5
Hasselt_0	Hasselt	Brussel	22	23	5	5
Hasselt_0	Brussel	Hasselt	24	25	0	5
Hasselt_1	Hasselt	Brussel	30	31	5	5
Hasselt_1	Brussel	Hasselt	32	33	0	5
Gent_1	Gent	Antwerpen	34.5	35.25	5	5
Hasselt_2	Hasselt	Brussel	36	37	5	5
Gent_1	Antwerpen	Gent	36.5	37.25	0	5
...	...	...	...	...	...	...

Table C.2: Sample output file **deliveries.csv**.

duetime	**startnode**	**endnode**	**start time**	**arrival time**	**transport time**	**handling time**	**hops**
3.01	Hasselt	Brussel	0.01	9.2	1	1.2	1
3.89	Hasselt	Brussel	0.89	9.4	1	1.4	1
3.89	Hasselt	Brussel	0.89	9.6	1	1.6	1
7.69	Hasselt	Brussel	4.69	9.8	1	1.8	1
9.75	Hasselt	Brussel	6.75	10	1	2	1
6.32	Gent	Antwerpen	3.32	10.45	0.75	1.2	1
11.73	Gent	Antwerpen	5.73	10.65	0.75	1.4	1
12.9	Hasselt	Brussel	8.9	11.2	1	1.2	1
13.68	Hasselt	Brussel	6.68	11.4	1	1.4	1
13.72	Hasselt	Brussel	8.72	11.6	1	1.6	1
13.87	Hasselt	Brussel	6.87	11.8	1	1.8	1
15.47	Hasselt	Brussel	8.47	12	1	2	1
12.78	Hasselt	Brussel	9.78	14.2	1	1.2	1
13.38	Hasselt	Brussel	9.38	14.4	1	1.4	1
14.56	Hasselt	Brussel	10.56	14.6	1	1.6	1
15.45	Hasselt	Brussel	10.45	14.8	1	1.8	1
18.78	Hasselt	Brussel	11.78	15	1	2	1
...	...	...	...	...	...	...	...

sample output file debug.log

```
0.00 | init_graph | initializing state graph...
0.00 | init_graph | state graph initialized
0.00 | order_generator Gent => Antwerpen | initializing incoming_containers...
0.00 | order_generator Gent => Antwerpen | incoming_containers initialized
0.00 | order_generator Gent => Antwerpen | activated
0.00 | order_generator Gent => Antwerpen | next arrival 27.06, last arrival 27.06
0.00 | order_generator Hasselt => Brussel | initializing incoming_containers...
0.00 | order_generator Hasselt => Brussel | incoming_containers initialized
0.00 | order_generator Hasselt => Brussel | activated
0.00 | order_generator Hasselt => Brussel | next arrival 1.05, last arrival 2.99
0.00 | order_generator Gent => Hasselt | initializing incoming_containers...
0.00 | order_generator Gent => Hasselt | incoming_containers initialized
0.00 | order_generator Gent => Hasselt | activated
0.00 | order_generator Gent => Hasselt | next arrival 1.95, last arrival 2.63
0.00 | Node Gent | initializing protocol...
0.00 | Node Gent | protocol initialized
0.00 | Node Gent | checking dispatch rules...
0.00 | Node Antwerpen | initializing protocol...
0.00 | Node Antwerpen | protocol initialized
0.00 | Node Antwerpen | checking dispatch rules...
0.00 | Node Brussel | initializing protocol...
0.00 | Node Brussel | protocol initialized
0.00 | Node Brussel | checking dispatch rules...
0.00 | Node Hasselt | initializing protocol...
0.00 | Node Hasselt | protocol initialized
0.00 | Node Hasselt | checking dispatch rules...
0.63 | order_generator Gent => Hasselt | add future arrival to incoming_containers
0.63 | order_generator Gent => Hasselt | next arrival 1.95, last arrival 3.70
0.99 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
0.99 | order_generator Hasselt => Brussel | next arrival 1.05, last arrival 3.50
1.00 | Node Gent | checking dispatch rules...
1.00 | Node Antwerpen | checking dispatch rules...
1.00 | Node Brussel | checking dispatch rules...
1.00 | Node Hasselt | checking dispatch rules...
1.05 | order_generator Hasselt => Brussel | generate container
1.05 | Node Hasselt | container arrived at intermediate hub
1.05 | order_generator Hasselt => Brussel | next arrival 1.10, last arrival 3.50
1.10 | order_generator Hasselt => Brussel | generate container
1.10 | Node Hasselt | container arrived at intermediate hub
1.10 | order_generator Hasselt => Brussel | next arrival 2.99, last arrival 3.50
1.50 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
```

```
1.50 | order_generator Hasselt => Brussel | next arrival 2.99, last arrival 4.62
1.70 | order_generator Gent => Hasselt | add future arrival to incoming_containers
1.70 | order_generator Gent => Hasselt | next arrival 1.95, last arrival 8.11
1.95 | order_generator Gent => Hasselt | generate container
1.95 | Node Gent | container arrived at intermediate hub
1.95 | order_generator Gent => Hasselt | next arrival 2.63, last arrival 8.11
2.00 | Node Gent | checking dispatch rules...
2.00 | Node Antwerpen | checking dispatch rules...
2.00 | Node Brussel | checking dispatch rules...
2.00 | Node Hasselt | checking dispatch rules...
2.62 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
2.62 | order_generator Hasselt => Brussel | next arrival 2.99, last arrival 7.48
2.63 | order_generator Gent => Hasselt | generate container
2.63 | Node Gent | container arrived at intermediate hub
2.63 | order_generator Gent => Hasselt | next arrival 3.70, last arrival 8.11
2.99 | order_generator Hasselt => Brussel | generate container
2.99 | Node Hasselt | container arrived at intermediate hub
2.99 | order_generator Hasselt => Brussel | next arrival 3.50, last arrival 7.48
3.00 | Node Antwerpen | checking dispatch rules...
3.00 | Node Brussel | checking dispatch rules...
3.00 | Node Gent | checking dispatch rules...
3.00 | Node Hasselt | checking dispatch rules...
3.50 | order_generator Hasselt => Brussel | generate container
3.50 | Node Hasselt | container arrived at intermediate hub
3.50 | order_generator Hasselt => Brussel | next arrival 4.62, last arrival 7.48
3.70 | order_generator Gent => Hasselt | generate container
3.70 | Node Gent | container arrived at intermediate hub
3.70 | order_generator Gent => Hasselt | next arrival 8.11, last arrival 8.11
4.00 | Node Antwerpen | checking dispatch rules...
4.00 | Node Brussel | checking dispatch rules...
4.00 | Node Gent | checking dispatch rules...
4.00 | Node Hasselt | checking dispatch rules...
4.62 | order_generator Hasselt => Brussel | generate container
4.62 | Node Hasselt | container arrived at intermediate hub
4.62 | order_generator Hasselt => Brussel | next arrival 7.48, last arrival 7.48
5.00 | Node Antwerpen | checking dispatch rules...
5.00 | Node Brussel | checking dispatch rules...
5.00 | Node Gent | checking dispatch rules...
5.00 | Node Hasselt | checking dispatch rules...
5.00 | Node Hasselt | send Truck Hasselt_0 to Node Brussel: possible to fill
5.00 | Truck Hasselt_0 | request forklift at Node Hasselt
5.00 | Truck Hasselt_0 | load cargo...
5.48 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
```

```
5.48 | order_generator Hasselt => Brussel | next arrival 7.48, last arrival 9.89
6.00 | Node Antwerpen | checking dispatch rules...
6.00 | Node Brussel | checking dispatch rules...
6.00 | Node Gent | checking dispatch rules...
6.00 | Node Hasselt | checking dispatch rules...
6.00 | Truck Hasselt_0 | finish loading cargo
6.00 | Truck Hasselt_0 | release forklift at Node Hasselt
6.00 | Truck Hasselt_0 | drive Hasselt => Brussel...
6.11 | order_generator Gent => Hasselt | add future arrival to incoming_containers
6.11 | order_generator Gent => Hasselt | next arrival 8.11, last arrival 16.13
7.00 | Node Antwerpen | checking dispatch rules...
7.00 | Node Brussel | checking dispatch rules...
7.00 | Node Gent | checking dispatch rules...
7.00 | Node Hasselt | checking dispatch rules...
7.00 | Truck Hasselt_0 | finish driving Hasselt => Brussel
7.00 | Truck Hasselt_0 | request forklift at Node Brussel
7.00 | Truck Hasselt_0 | unload cargo...
7.20 | Node Brussel | container arrived at final destination
7.40 | Node Brussel | container arrived at final destination
7.48 | order_generator Hasselt => Brussel | generate container
7.48 | Node Hasselt | container arrived at intermediate hub
7.48 | order_generator Hasselt => Brussel | next arrival 9.89, last arrival 9.89
7.60 | Node Brussel | container arrived at final destination
7.80 | Node Brussel | container arrived at final destination
7.89 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
7.89 | order_generator Hasselt => Brussel | next arrival 9.89, last arrival 9.99
7.99 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
7.99 | order_generator Hasselt => Brussel | next arrival 9.89, last arrival 11.91
8.00 | Node Antwerpen | checking dispatch rules...
8.00 | Node Brussel | checking dispatch rules...
8.00 | Node Gent | checking dispatch rules...
8.00 | Node Hasselt | checking dispatch rules...
8.00 | Node Brussel | container arrived at final destination
8.00 | Truck Hasselt_0 | finish unloading cargo
8.00 | Truck Hasselt_0 | release forklift at Node Brussel
8.00 | Node Brussel | Truck Hasselt_0 available
8.11 | order_generator Gent => Hasselt | generate container
8.11 | Node Gent | container arrived at intermediate hub
8.11 | order_generator Gent => Hasselt | next arrival 16.13, last arrival 16.13
9.00 | Node Antwerpen | checking dispatch rules...
9.00 | Node Brussel | checking dispatch rules...
9.00 | Node Brussel | send Truck Hasselt_0 to Node Hasselt: not our truck
9.00 | Truck Hasselt_0 | drive Brussel => Hasselt...
```

```
9.00 | Node Gent | checking dispatch rules...
9.00 | Node Hasselt | checking dispatch rules...
9.89 | order_generator Hasselt => Brussel | generate container
9.89 | Node Hasselt | container arrived at intermediate hub
9.89 | order_generator Hasselt => Brussel | next arrival 9.99, last arrival 11.91
9.91 | order_generator Hasselt => Brussel | add future arrival to incoming_containers
9.91 | order_generator Hasselt => Brussel | next arrival 9.99, last arrival 12.13
9.99 | order_generator Hasselt => Brussel | generate container
9.99 | Node Hasselt | container arrived at intermediate hub
9.99 | order_generator Hasselt => Brussel | next arrival 11.91, last arrival 12.13
10.00 | Node Antwerpen | checking dispatch rules...
10.00 | Node Brussel | checking dispatch rules...
10.00 | Truck Hasselt_0 | finish driving Brussel => Hasselt
10.00 | Node Hasselt | Truck Hasselt_0 available
...
```