Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Network Delay Model Creation and Validation for Design Space Exploration of Distributed Cyber-Physical Systems

Author: William Ford (2712009)

1st supervisor:	Benny Akesson
daily supervisor:	Faezeh Sadat Saadatmand
2nd reader:	Zhiming Zhao

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

November 12, 2024

Abstract

In recent years, large-scale distributed cyber-physical systems (dCPS) have become the driving force behind world-class manufacturing companies like ASML, Canon Production Printing, and Philips. However, the task of evaluating the designs of these systems by building physical prototypes has grown in complexity and cost. Automated scalable Design Space Exploration (DSE) offers a promising solution to this problem. Using automated processes to evaluate dCPS design alternatives significantly reduces the time and cost of the design process. However, DSE of dCPS brings its own challenges.

This thesis aims to enhance the design process of dCPS as part of the DSE2.0 research project, specifically addressing the challenge of modeling the delay of the network that connects the dCPS subsystems. The goal is to model network delays at an appropriate level of abstraction such that the model has reasonable speed and accuracy, and is useful for DSE purposes.

The research methodology encompasses the formalization of the concepts of Network Topology and Network Traffic, establishing an open-source framework called GeNSim for representing and generating such data. This is followed by the proposal of four network delay models at different levels of abstraction: Constant Delay, Constant Bandwidth, Latency-Rate, and INET. The models are evaluated for accuracy by means of a case study using real-world data from an ASML TwinScan test bench machine, and for simulation speed using synthetic data generated by GeNSim.

The experimental evaluation demonstrates that each model has a different set of strengths and weaknesses, and none of the models pass all speed, accuracy, and usefulness requirements simultaneously, demonstrating an apparent tradeoff.

Based on these findings, we propose a multi-step modeling approach, leveraging the strengths of multiple models and canceling out their weaknesses.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

In recent years, Distributed Cyber-Physical Systems (dCPS) have emerged as a cornerstone of innovation across many industries, including healthcare, industrial automation, robotics, and avionics. These dCPS comprise heterogeneous multi-core or many-core systems connected through complex networks (1). As these systems are pushed to reach new technological heights, their increasing complexity introduces significant challenges. Traditional approaches of evaluating dCPS designs, such as constructing physical prototypes, are often impractical due to the high complexity, costs and time consumption associated with an iterative design processes.

Automated Design Space Exploration (DSE) for dCPS has gained attention as a promising approach to address this challenge. DSE refers to the process of (automatically) searching the vast *design space* of all possible dCPS configurations for one or more configurations that best satisfy the given *design objectives*. This process can significantly reduce dCPS design costs and efforts.

However, DSE in the context of dCPS poses several critical challenges. First, the complex nature of these systems creates a design space with an extremely large number of possible configurations to evaluate. If the assessment of these design points is conducted via a simulation model, even achieving marginal reductions in execution time can yield large cumulative gains. Second, in order to evaluate a design point, the simulation model needs to be able to capture both the computing hardware components and the software components of the system. In order to do this, one essential part of the simulation model is the network model, which must accurately represent the delays and complex interactions in the network connecting the dCPS subsystems.

Network delays can drastically affect the performance of dCPS, making their accurate modeling essential for evaluating potential design configurations using DSE. Current

methodologies lack a comprehensive approach that fully addresses the nuances of network topology and traffic within a dCPS. Previous research has primarily focused on exploring resource-sharing abstractions for single resources like memory controllers, but have not extended these methods to the complex, interconnected networks found in dCPS.

This gap highlights the need for network delay models that are both accurate and computationally efficient. Specifically, models should be capable of capturing the nuances of network topology and traffic within dCPS while being suitable for rapid evaluation of numerous design points. Moreover, they should allow for the exploration of different network configurations, such as changes in topology or bandwidth, which is crucial in DSE.

DSE2.0 (1) is an ongoing research project that aims to extend the state-of-the-art in DSE, to find ways to overcome the aforementioned challenges, and to leverage the DSE process for large industrial dCPS like the ASML TwinScan machine. This thesis aims to support this research project by comprehensively proposing, developing, and validating a network delay model, that is suitable to be used in the context of DSE of dCPS. The goal is to find a model that strikes an appropriate balance between accuracy (how well the model is able to replicate the real-world system), speed (how long it takes to evaluate design points), and general usefulness (how well the model is able to adapt to new system configurations).

The contributions of this thesis are threefold:

1. **Formalization of network topology and traffic framework**: This work establishes a conceptual framework for network topology and network traffic called GeNSim. This framework involves a description of which real-life elements of network topologies and network traffic are involved in the model and which are not, laying a foundation for representing any network delay model and its traffic in terms of this framework. It also details methods for automatic generation of these data structures.

2. **Proposal of four network delay models**: Leveraging the established framework, the thesis proposes four network delay models at different levels of abstraction: the Constant Delay model, the Constant Bandwidth model, the Latency-Rate model, and the INET model. Each model offers a different trade-off between accuracy and computational complexity. By evaluating models across this spectrum, we aim to identify an approach that is both accurate enough to provide meaningful insights and fast enough to be practical for DSE.

3. **Experimental evaluation**: To validate the applicability of the proposed models, each model is evaluated based on two key performance indicators: modeling accuracy and simulation speed. We assess their accuracy using real-world data from an industrial case study involving an ASML TwinScan Test Bench, providing insight into how well the models capture actual network behavior. Additionally, we evaluate their simulation speed and scalability using synthetic data generated by GeNSim. This dual evaluation favours a model that is reasonably accurate and fast, and therefore potentially practically applicable in real-world dCPS design scenarios.

The research presented in this thesis is structured as follows. Chapter 2 provides the necessary background information to understand the concepts in this report. Chapter 3 reviews the previous work related to this thesis. Chapter 4 outlines the methodology used in this research. In Chapter 5, we describe the formalization of network topologies and traffic and introduce the GeNSim framework. Chapter 6 details the proposed network delay models. Chapter 7 presents the experimental evaluation and results. Chapter 8 discusses the findings of the previous chapters. Finally, Chapter 9 concludes the thesis with a summary of the contributions and suggestions for future work.

2

Background

This chapter provides the basic concepts required to understand this thesis. We begin with an introduction to Design Space Exploration (DSE) and its relevance to distributed cyber-physical systems (dCPS). We then delve into network modeling, and how this relates to discrete-event simulation. Finally, we discuss network model validation techniques and the related practical considerations.

2.1 Design Space Exploration

Design Space Exploration (DSE) is the process of systematically evaluating a space of candidate design solutions, called *design points*, that satisfy a given set of design objectives (1). In the context of complex systems, DSE aids designers in making informed decisions by providing insights into trade-offs between different design parameters. The DSE process typically consists of four main stages, as illustrated in Figure 2.1):

1. **Modeling**: the system is discovered, described, abstracted, and mapped to a representation of the system that captures essential characteristics while omitting unnecessary details; a model.

2. **Design space creation**: based on the design choices, a design space of all possible design points on the model is spanned. It is common practice to perform a preliminary pruning phase, removing design points that are invalid in ways that are easy to detect.

3. **Design space exploration**: a search algorithm evaluates the available design points in the design space. Some examples of search algorithms are exhaustive search and

heuristic-based (e.g. evolutionary algorithms or simulated annealing) search. The algorithm may dynamically prune design points based on evaluation results.

4. **Results**: the outputs of a DSE process are intermediate outcomes or conclusive design recommendations.

DSE has been successfully adopted in several areas, such as low-level hardware design for Systems-on-a-Chip (SoC) (3) and Multiprocessor System-on-a-Chip (MPSoC) (4). However, the growing complexity of distributed Cyber-Physical Systems (dCPS) poses several challenges for DSE, such as combinatorial explosion and simulation scalability.

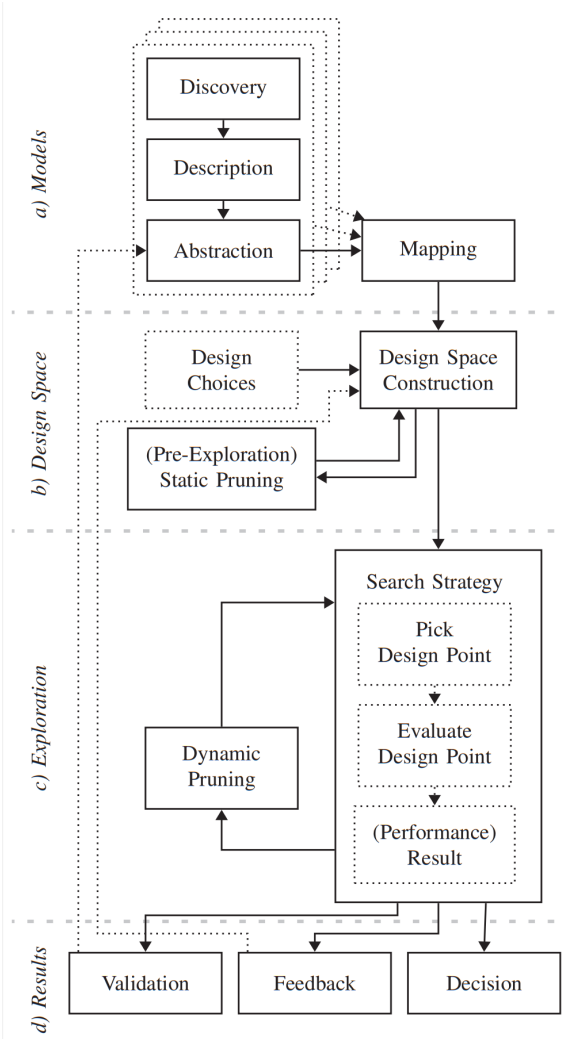

Figure 2.1: The general Design Space Exploration workflow (1).

2.2 Distributed Cyber-Physical Systems

Distributed Cyber-Physical Systems (dCPS) are systems that integrate computational elements with physical processes, where the computational elements are distributed across multiple devices that interact through complex internal networks. Examples of dCPS include industrial automation systems, autonomous vehicles, smart grids, and advanced manufacturing equipment like the ASML TwinScan lithography machines. Key characteristics of dCPS include:

- **Heterogeneity**: dCPS consist of a diverse set of computing and physical components, including processors, sensors, actuators, and communication networks, often from different vendors and with varying capabilities.
- **Concurrency**: Multiple components operate concurrently, interacting and exchanging information in real-time.
- **Timing Constraints**: Many dCPS applications are time-critical, requiring precise synchronization and meeting strict timing deadlines.
- **Scalability**: dCPS often need to scale to accommodate more components or higher performance requirements.

Designing dCPS is challenging due to the need to manage complex interactions between computational and physical components, ensure reliability and safety, and meet performance requirements. While DSE offers a structured approach to navigate the complex design space of dCPS, several challenges hinder its effective application. Firstly, the heterogeneity and scale of dCPS result in an enormous design space with a combinatorial explosion of possible configurations. Evaluating each design point is computationally intensive, making exhaustive exploration impractical. Secondly, accurately modeling both the hardware and software components of dCPS is essential for reliable evaluation. This includes processors, memory systems, sensors, actuators, and importantly, the communication network. High-fidelity models provide accurate results but are computationally expensive, leading to long simulation times. Conversely, abstract models run faster but may lack sufficient fidelity to be accurate.

2.3 Network Modeling

Network modeling involves the abstraction of communication network features and properties, enabling the creation of analytical representations with varying levels of complexity (5). There are various types of network modeling approaches, including: stochastic modeling, analytical modeling, and simulation modeling.

Stochastic models assume the system evolves randomly over time according to some stochastic process. If the system state X_n is observed at discrete time points $n = 0, 1, 2, \ldots$, we say that $X_n, n \geq 0$ is a *discrete-time* stochastic process. If the system state $X(t)$ is observed continuously in time, it is described by a *continuous-time* stochastic process $X(t), t \geq 0$. It is a common assumption in stochastic modeling of network traffic that packets arrive at the server according to a Poisson process. A Poisson process is a continuous-time stochastic counting process of which the inter-arrival times follow an exponential distribution and are Independent and Identically Distributed (IID) (6). Due to this assumption of arrivals that follow a Poisson process, discrete-event dynamic systems are often modeled using continuous-time versions of the stochastic processes (5). While stochastic models are valuable for understanding general network behavior under random processes, they may not be ideal for DSE of dCPS. Stochastic models often rely on assumptions like Poisson arrivals, which may not accurately reflect the specific and deterministic traffic patterns in dCPS. Moreover, they can become mathematically complex and less tractable for large-scale systems. Some popular examples of stochastic modeling techniques are Markov Chains (6), Queuing Networks (7), and Petri Nets (8).

Analytical models use mathematical equations to represent network behavior. They may be used in a stochastic context if necessary, but they are not inherently stochastic. Rather they have parameters that can be tuned to achieve results that are similar to the behaviour of the real-life system. An example of this is the Latency-Rate servers abstraction, which is commonly used to analyze traffic scheduling algorithms. The model is characterized by two parameters: the maximum server latency Θ_i and the minimum service rate ρ_i. The service rate represents the guaranteed rate at which the server processes packets sent by the client, while the latency represents the maximum time until that rate can be guaranteed (9). This guaranteed service to a client is independent of the service requests from other clients, and is achieved by the use of accounting (reserving resources) and enforcement (no more service when the budget is depleted) (2). A *system busy period* is a maximal interval of time during which the server is never idle (9). Figure 2.2 shows an example of how a Latency-Rate server might react to incoming bursts of traffic (2).

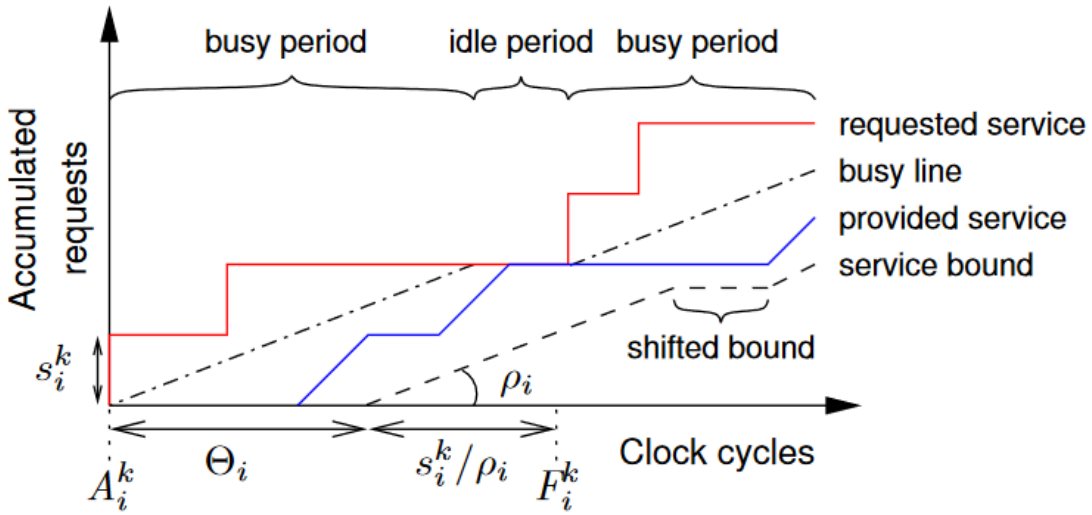

Figure 2.2: A Latency-Rate server and its associated concepts (2).

Simulation models use computer simulation software to evaluate network behavior. The de facto standard of network simulation is discrete-event simulation, which is a technique for modeling the behavior of a complex system as a sequence of discrete events that occur over time. The number of events is finite, and can include events such as messages exchanged between system components, or state transitions (10). There are two basic types of discrete-event simulation: trace-driven simulation, where the simulation inputs come from data captured on the real system, and and stochastic simulation, where the workload is characterized by probability distributions (5). Events in a trace-driven discrete-event simulation are usually defined in terms of their occurrence time, duration, and impact on the system state. The state of the system is represented by a set of variables that capture important aspects of its behavior, such as the number of entities in the system, the status of resources, or the progress of a particular process. At each event, the simulator determines the impact of the event on the system state, updates the relevant variables, and schedules any future events that may be triggered by the current event (10).

Discrete-event simulations have become a popular technique for modeling complex computer networks and for analyzing their behavior. One popular implementation is the INET framework for OMNeT++, an open-source network simulation library that is able to model various network protocols and emulate network hardware components (11). Another example is NS-3, an open-source discrete-event network simulator designed for research and educational purposes. It provides detailed models for network protocols and supports simulation of complex network topologies (12).

Discrete-event simulations are valuable tools for understanding the complex interactions between components in a dCPS and for guiding dCPS design decisions. However, there are several research gaps and challenges related to discrete-event simulation for dCPS design space exploration. One limitation is the scalability of discrete-event simulations for large-scale and complex dCPS, as the number of events, interactions, and components increases (13). Another challenge is the lack of standardized modeling approaches and simulation tools for dCPS. While commercial tools such as NetSim, OPNET, and QualNet provide a comprehensive set of features for simulating computer networks, open-source simulators such as OMNeT++, NS-2, NS-3, and J-SIM offer platforms that are extensible for researchers and developers to experiment with non-standard network configurations and protocols.

2.4 Model Validation

Model validation is often defined as the substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model (14). More specifically, we will mainly be focusing on model scenario validation, which refers to the process of proving that a reasonable relationship exists between the simulation experiments and the real life situations in which the corresponding technology will be deployed (15). Providing this validity substantiation is a critical step in the process of developing a (simulation) model, as models based on hypothetical relationships, faulty code or incorrect data are void of meaning. Still, model validation is known as one of the most challenging methodological issues associated with computer simulation techniques, because there are many approaches and not one given plan that works for every project (16).

One commonly accepted method to validate a simulation model is the Naylor and Finger validation approach (16), which includes three phases: Face Validity (asking system experts whether the model behavior is reasonable), Validation of Model Assumptions, and Validation of Input-Output transformations (comparing the real system's and the model's outputs using the same input data) (17). This is also known as Multistage validation, combining the three historical methods of Rationalism, Empiricism and Positive Economics. Additional validation methods may include: Historical Data Validation (using part of the historical system data to build the model, and the remaining data to test if the model behaves as the system does to avoid overfitting the model), Extreme-Condition Testing (whether the model output is plausible for extreme and unlikely conditions), and Turing

Tests (system behavior experts are asked if they can discriminate between system and model outputs) (14). Turing tests by expert opinion are a popular way to incrementally validate a model (5).

Many of these methods aim to validate a simulation model by comparing its output to real-world data. To obtain this real-world data it must be collected and empirically verified. Data collection in the context of network modeling for dCPS begins with the step of network exploration, in which the topology of the network is automatically discovered and registered. It is also possible to do this manually, but effort is reduced by a great deal by employing automatic network discovery due to the complex nature of dCPS networks. Following that step comes network measurement, which deploys tracing software on the topology to measure network logs and performance metrics. These logs may be used to drive trace-driven discrete event simulations, and then to compare with the simulation outputs.

However, network exploration and measurement suffer from a similar issue as discrete-event simulations: there is a severe lack of standardized approaches and tools. Available network exploration tools mainly differ in the level of automation and comprehensiveness. Tools such as Advanced IP Scanner, LanTopoLog, and NetScanTools are primarily used for port scanning, host discovery, and network mapping, while Open-AudIT, Device42, and Total Network Inventory offer more comprehensive network discovery, inventory management, and IT documentation capabilities. Network measurement tools can be broadly categorized into two categories: monitoring platforms and distributed tracing platforms. Monitoring platforms, such as Datadog, New Relic, and Netdata provide observability across infrastructure, applications, and logs, while distributed tracing systems, such as Jaeger, and Zipkin, visualize the timing and duration of requests in complex distributed systems. Some tools, such as AppDynamics and Dynatrace, offer both monitoring and distributed tracing capabilities.

3

Related Work

This chapter reviews the existing literature related to network modeling, particularly focusing on its application to the context of DSE for dCPS. The discussion progresses from general network delay models to specific studies that have explored resource abstractions and their relevance to DSE. Finally the research gap this thesis aims to address is described.

Network delays have been modeled for a wide range of applications beyond DSE. Section 2.3 discusses some high-level approaches and some examples of models that adhere to those approaches. Out of these approaches, building a simulation model of the entire network topology seems to be the de facto standard for practical and industrial use cases of network delay modeling. Though, the simulation models that result from this approach, while highly accurate, are impractical for DSE due to their slow execution times. Since the DSE process involves evaluating a potentially massive number of possible network configurations, a model that is effective in this use case must strike a balance between accuracy and speed.

This phenomenon of the speed-accuracy tradeoff is not limited to just network modeling. (18) provides an in-depth analysis of SDRAM interference in memory controllers, comparing a detailed implementation with Latency-Rate server abstractions. Their research demonstrates the efficacy of Latency-Rate abstractions in simplifying the analysis of memory controller behavior while maintaining accuracy. This study illustrates the potential of Latency-Rate abstractions for single-step resources like memory controllers, emphasizing the importance of abstraction in reducing computational complexity. However, their focus remains on a single resource type, the memory controller, without extending to the interconnected nature of network resources. The distinction in this thesis lies in its focus on the distributed and complex nature of network resources in distributed Cyber-Physical Systems (dCPS), an area not addressed in their study.

(19) extends the application of Latency-Rate servers by including the modeling of shared resources in real-time streaming applications on a multi-processor system-on-chip (MP-SoC). Their work explores CPU sharing, network-on-chip (NoC) sharing, and memory sharing within an FPGA-based system. It shows how Latency-Rate servers can be used as analysis models for these shared resources, though primarily in a data flow model rather than a simulation model, and their scope does not include network-wide abstractions. Besides, this work is making models of the worst-case timing behavior to be able to compute guaranteed minimum throughput. The latency and rate parameters are hence computed analytically based on intimate knowledge of all hardware. This work builds on these insights by developing a simulation model that leverages Latency-Rate abstractions for network resources that infers the parameters from the available data and they correspond to typical, rather than worst case, network behavior.

(1) reviews the state-of-the-art in DSE for dCPS. They argue that efficient and scalable DSE technology for dCPS is largely non-existent and represents a significant research gap. The paper identifies several scientific challenges, such as the need for scalable modeling techniques and efficient exploration of vast design spaces. The authors emphasize that current DSE methods, often used in Systems-on-Chip (SoC) and Multi-Processor Systems-on-Chip (MPSoC) designs, are insufficient for the more complex and heterogeneous nature of dCPS.

A notable point in their review is the use of SESAME, a trace-driven simulation and modeling framework for embedded MPSoC systems. SESAME allows for the evaluation of design points using high-level abstractions, and supports the exploration of different architectural configurations through a combination of genetic algorithms and discrete event simulation. Despite its utility in MPSoC domains, the application of SESAME to network-wide DSE in dCPS remains limited. This limitation highlights the necessity for extending such frameworks to accommodate the unique requirements of dCPS, particularly in terms of network modeling and scalability.

In summary, while significant literature has been written on network modeling and the application of Latency-Rate servers for resource analysis, there remains a critical gap in their application to DSE for dCPS. The existing research typically focuses on single resource types or small-scale embedded systems. This thesis aims to bridge this knowledge gap by developing network delay models at different levels of abstraction, and evaluating them on the basis of their applicability to the large-scale, complex nature of DSE in dCPS, contributing a novel solution to this open area of research.

4

Methodology

The primary objective of this research is to evaluate the suitability of a given set of network delay models for Design Space Exploration (DSE) of distributed cyber-physical systems (dCPS). Given the inherent subjective nature of *suitability*, this study attempts to quantify this concept in terms of three distinct metrics:

- **Accuracy**: the extent to which the estimated delays of the target model compare to the real-world behaviour of the modeled system.
- **Speed and Scalability**: the time it takes for a simulation of the target model to complete and how this scales as a simulation grows in complexity.
- **Usefulness**: whether the target model is able to capture the core DSE activities of altering: (1) the network bandwidth, (2) the network latency, (3) both of these at the same time, or (4) neither of these.

This research is conducted in three broad, consecutive steps, each one corresponding to one of the contributions of this thesis. Figure 4.1 illustrates the elements of, and dependencies between the contributions of this work as a high-level flow chart.

The first step, as will be discussed in Chapter 5, involves creating an abstraction of the concepts of network topologies and network traffic. This abstraction process is a common step in modeling, and helps with identifying the key concepts that are relevant to the modeling domain. Once this framework is established and it is clear which concepts are included and excluded from our abstract representation, it can be applied and tested. First, it is applied to real-world networking data from an industrial case study. This not only validates the framework's capability to capture the relevant data, but also provides a basis for assessing the accuracy metric. Second, a method for generating synthetic instances of

the data is proposed. This synthetic data is employed to evaluate the speed and scalability metrics of the models, testing the scenarios for which real-world data is not available.

The second step, further discussed in Chapter 6, is dedicated to defining the four proposed network delay models that will be evaluated in this research. These models represent a spectrum of abstraction, ranging from a network delay model that is very abstract, to one that captures many fine details. Discussing the core ideas behind each of the models enables an evaluation of their usefulness in supporting the aforementioned DSE activities.

The final step of this process, as discussed in Chapter 7, involves conducting two experiments on the network delay models. The first experiment measures model accuracy by comparing the model output with that from the real-world industrial case study. The second experiment measures the models' speed and scalability by leveraging synthetic networking data to determine how well each model scales in terms of simulation speed as the workload increases.

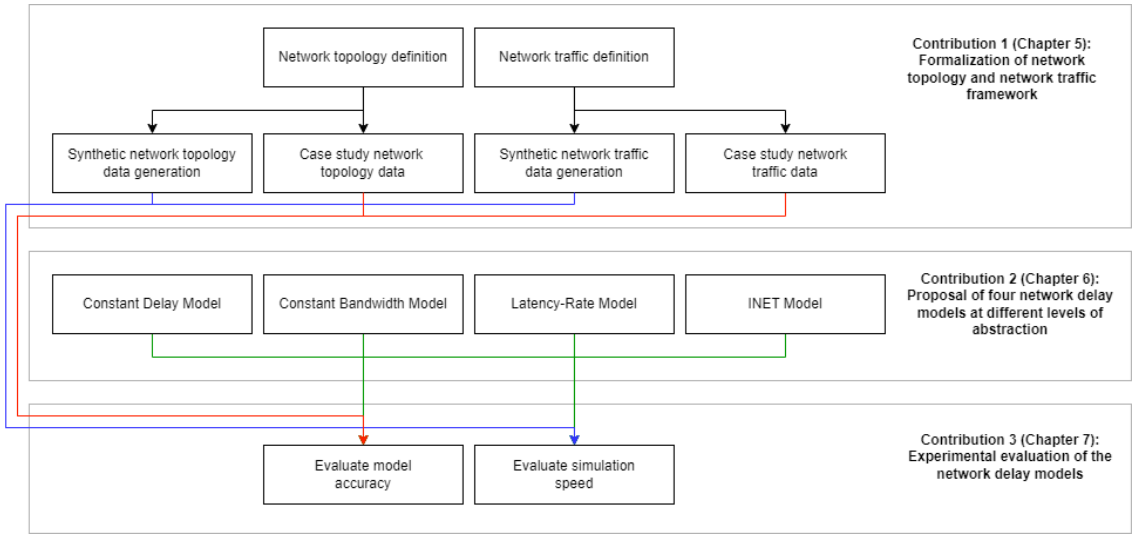

Figure 4.1: A flow diagram that illustrates the dependencies between the contributions of this thesis. The blue arrows represent synthetic data, the red arrows represent real-world data, and the green arrows represent models that interpret this data and model the network delay based on it.

5

Network Topologies and Traffic

In order to evaluate a network delay model, one must first be able to define network topologies and network traffic in a reproducible manner. This section details the design of GeNSim (**Gen**erate **N**etwork **Sim**ulations), an open-source data framework and Python Command Line Interface (CLI) application developed for this research project to represent network topologies and traffic in a standardized format.

GeNSim supports representing networking data by printing directly to the console, or by outputting a JSON file, XML file, or an OMNeT++ simulation. Additionally, GeNSim can convert networking data between its various output formats and validate them to ensure they conform to the GeNSim data model. By making GeNSim open-source, subsequent research projects may use its functionalities, with the repository currently available on GitHub at https://github.com/quintal-william/gensim.

5.1 Data Model

In GeNSim, a network topology is defined as a graph of nodes. A node may be one of two types: a leaf node or a topological node. Leaf nodes represent the smallest possible communicating entities (e.g. end-hosts, switches). Topological nodes, on the other hand, represent sub-topologies, which are themselves graphs of nodes. This hierarchical structure allows for the easy definition of complex structures made up of nested topologies. Connections between leaf nodes are represented by edges, which are uni-directional. However, in practice, edges are typically made bidirectional by creating one for each direction. Figure 5.1 illustrates an example of the internal representation of a simple topology of two leaf nodes connected by a 1 Gbps edge, which is also visually represented in Figure 5.2.

```
{
  "type": "topology",
  "id": "ExampleMesh",
  "nodes": [
    {
      "type": "host",
      "id": "ExampleMesh_0",
      "edges": [
        {
          "type": "edge",
          "id": "ExampleMesh_0_ExampleMesh_1_1000000000",
          "source": "ExampleMesh_0",
          "destination": "ExampleMesh_1",
          "bandwidth": 1000000000
        }
      ]
    },
    {
      "type": "host",
      "id": "ExampleMesh_1",
      "edges": [
        {
          "type": "edge",
          "id": "ExampleMesh_1_ExampleMesh_0_1000000000",
          "source": "ExampleMesh_1",
          "destination": "ExampleMesh_0",
          "bandwidth": 1000000000
        }
      ]
    }
  ]
}
```

Figure 5.1: A simple topology of two leaf nodes connected by a 1 Gbps edge as represented by GeNSim in JSON format, visually represented in Figure 5.2.

Network traffic in GeNSim is defined as a list of arrivals. Arrivals represent messages of a certain size that enter the network at a specified time, flowing through the network

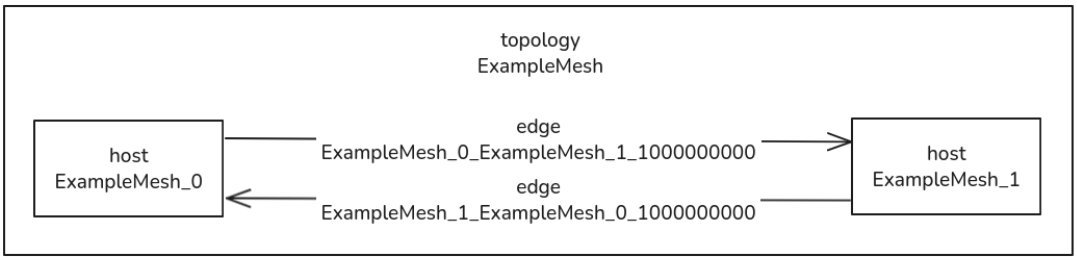

Figure 5.2: A visual representation of the GeNSim JSON representation of a simple two-node topology given by Figure 5.1

from a source to the intended destination. Figure 5.3 provides an example an internal representation of a network traffic trace corresponding to the network topology shown in Figure 5.1.

5.2 Synthetic Data

In order to generate synthetic instances of the data in the defined format, the concept of *connectedness* is introduced. The connectedness variable C refers to the probability that two leaf nodes are bi-directionally connected. This logic extends to connecting two topological nodes as well: C is the probability that two leaf nodes are connected for each combination of leaf nodes in the to-be-connected topological nodes. Thus, any two nodes may be connected according to a connectedness probability C, where $C = 1$ means all leaf nodes are fully interconnected, and $C = 0$ implies no connections between any pair of leaf nodes. Intermediate values represent varying degrees of connectedness.

This parameter C creates opportunities to randomly generate a variety of potentially complex network topologies. GeNSim includes two built-in topology generators, and allows to easily define custom topology generators:

1. **Mesh**: Interconnects a number of nodes according to a given connectedness parameter C.

2. **Star**: Connects a number of nodes to a central node according to a given connectedness parameter C.

In addition to these standard topologies, various combinations can be experimented with. As an example, creating a star topology with $C = 0.5$, with as a central node a

```
{
  "type": "traffic",
  "id": "ExampleMesh-Traffic",
  "arrivals": [
    {
      "type": "arrival",
      "id": "0.00971_ExampleMesh_0_ExampleMesh_1_974",
      "time": 0.00971,
      "source": "ExampleMesh_0",
      "destination": "ExampleMesh_1",
      "size": 974
    },
    {
      "type": "arrival",
      "id": "0.02208_ExampleMesh_1_ExampleMesh_0_580",
      "time": 0.02208,
      "source": "ExampleMesh_1",
      "destination": "ExampleMesh_0",
      "size": 580
    }
  ]
}
```

Figure 5.3: A short network traffic definition applied to the topology shown in Figure 5.1 as represented by GeNSim in JSON format.

mesh topology with $C = 1$, results in the leaf nodes of the star topology connecting to, on average, half of the central nodes, which are internally fully connected.

Once a topology is generated and saved to a file, the next step is to generate traffic for it. GeNSim provides three methods to generate network traffic, with the option to implement custom generators:

1. **Constant interval**: Assigns a uniform value to each inter-arrival time.
2. **Poisson distribution**: Draws inter-arrival times from an exponential distribution. All arrivals are independent.
3. **Packet train model**: Creates "packet trains", which are bursts of packets with a given maximum train length, and draws the time between "cars" of the train and the

time between trains from two different exponential distributions.

Existing network modeling tools often lack the flexibility or specificity required for dCPS modeling. For instance, many tools do not support hierarchical topologies or the generation of synthetic traffic that accurately reflects dCPS characteristics. GeNSim addresses these gaps by providing a customizable framework that can represent hierarchical structures through topological nodes and leaf nodes, allowing for the modeling of complex, nested network architectures common in dCPS. Additionally, its traffic generation capabilities, including support for different distribution models and packet train behaviors, enable the creation of realistic synthetic workloads. This flexibility makes GeNSim a novel contribution that fills a critical need in network delay modeling for DSE of dCPS.

6

Network Model Creation

This section provides a description of the four network delay models that are evaluated in this thesis. These models were chosen to span the abstraction spectrum, first discussing a very abstract model, incrementally moving to more detailed models, ending at a model that captures many network details. For each model, it is discussed how it uses its parameters to approximate network delays. For models that are defined by an equation, the equation is formulated in the following terms: ω is the vector of all messages contained in a network traffic trace, t_a, t_f, and s are the functions that represent the arrival time (in seconds), finishing time (in seconds), and size (in bytes) of a given message, respectively. As an example, $t_a(\omega^k)$ refers to the arrival time of the k^{th} message in the network traffic trace ω. Furthermore, each network model's usefulness for DSE of dCPS is discussed in terms of whether the target model is able to capture the four core DSE activities discussed in Section 4. By evaluating these models with different levels of abstraction, one model could be chosen that is appropriately abstract such that is has reasonable speed, accuracy, and usefulness for the purpose of DSE for dCPS.

6.1 Constant Delay Model

Starting at the most abstract end of the spectrum, the Constant Delay model always adds the same delay to any message coming into the system. This delay parameter Θ is the only parameter of this model, and it is set at the start of a simulation. The Constant Delay model is non-blocking, meaning that it may handle any number of network messages simultaneously. Therefore, it can be represented by the Equation 6.1, which calculates the finishing time of the k^{th} network message $t_f(\omega^k)$ by adding the arrival time $t_a(\omega^k)$ to the constant delay Θ.

$$t_f(\omega^k) = t_a(\omega^k) + \Theta \tag{6.1}$$

In the context of DSE for dCPS, the Constant Delay Model is most useful in the case of wanting to add or remove hops to the network topology, altering the mean network latency in the process. This example is represented by usefulness case 2. Besides that, it may also be used for case 4, in which there are no changes made to the network.

6.2 Constant Bandwidth Model

The Constant Bandwidth Model approximates the delay of the k^{th} network message in trace ω, by dividing the message size $s(\omega^k)$ by a bandwidth ρ in bytes per second, as shown in Equation 6.2. The message size will be provided dynamically during the simulation based on the defined network traffic trace. The bandwidth ρ is a static parameter that is set at the start of a simulation.

$$t_f(\omega^k) = t_a(\omega^k) + \frac{s(\omega^k)}{\rho} \tag{6.2}$$

Similarly to the Constant Delay model, the Constant Bandwidth model is non-blocking by design. Furthermore, one could argue that it is less abstract than the Constant Delay model, because it takes the size of the network message into account. It is therefore most fit for the usefulness cases 1 and 4, in which either the bandwidth of the network is altered, or nothing at all.

6.3 Latency-Rate Model

The Latency-Rate Model is based on the Latency-Rate servers abstraction, as discussed in Section 2.3. The Latency-Rate servers abstraction uses Equation 6.3 to calculate the term $t_f(\omega^k)$, which represents the **worst-case** finishing time of the k^{th} network request. We, however, will use the equation to approximate the **actual** network delay.

$$t_f(\omega^k) = max(t_a(\omega^k) + \Theta, t_f(\omega^{k-1})) + \frac{s(\omega^k)}{\rho} \tag{6.3}$$

The Latency-Rate Model combines Equations 6.1 and 6.2, resulting in a model with two parameters: the latency Θ, and the rate ρ. These parameters, similarly to those of the other analytical models, are fixed, and supplied at the start of a simulation. Besides that,

Equation 6.3 introduces a "max" clause, ensuring that $t_f(\omega^k)$ does not preceed $t_f(\omega^{k-1}) + \frac{s(\omega^k)}{\rho}$, which makes it a blocking model.

Owing to its dual parameter setup, the Latency-Rate server is able to work with all four usefulness scenarios. If a hop is added to the network, the latency parameter is altered, if a link changes in bandwidth, the rate parameter is tweaked. This also extends to the scenarios in which none or both of these are updated.

6.4 INET Model

The INET Model is vastly different from the other three models discussed in this section. It is not represented by a single calculation, rather, it leverages the OMNeT++ INET framework to simulate a part-by-part replica of the network topology that is being modeled. The INET model is able to simulate complex network interactions with a high level of detail. It is therefore the least abstract of the four network delay models considered in this thesis, and the most difficult to set up. It requires intimate knowledge of a given network topology. This data may be collected manually, or using automated network crawlers such as OpenAudIT (20). Besides, it needs a detailed view of all interfering traffic to be considered accurate.

In terms of usefulness, the INET Model is able to cover all four DSE of dCPS use-cases. Since it simulates the entire topology and all its network interactions, making changes to the topology is as trivial as just changing the in-model topology to reflect the newly desired topology. Besides, since the INET Model does not approximate the delay using an equation, it has no parameters, which makes it simpler to use. These characteristics make it highly useful for DSE of dCPS.

7

Experimental Evaluation

This chapter documents the experimental evaluation of the models defined in Chapter 6 according to the network topology and traffic framework described in Chapter 5. This evaluation is twofold, consisting of one experiment to evaluate the models' accuracy using data from an industrial case study, and another experiment to evaluate the models' speed using generated network traffic and topology data. This distinction in data source exists because the former experiment requires comparing the model output to a real-world data stream, while the latter experiment requires testing the models against arbitrary topologies and traffic patterns, where obtaining real-world data is highly difficult and largely irrelevant to the results of the experiment. Evaluating especially these two factors is crucial for determining which model is most suitable for DSE of dCPS, striking the balance in the speed-accuracy tradeoff.

7.1 Implementation

This section describes the experimental setup and implementation details for the model evaluation process. The setup includes a custom, model-agnostic simulation framework, allowing for the evaluation of each of the four network delay models defined in Section 6. Additionally, this section outlines the system configuration used for running the simulations in detail.

7.1.1 Simulation Setup

The experiments are conducted using the discrete event simulator OMNeT++. Within this simulation software, a custom lightweight simulation framework was designed to aid the experimentation process. As shown in Figure 7.1, this framework consists of three

core components that connect to the network model and control the flow of messages: the network *entry*, the network *exit*, and the network *analyzer*. A message, in this context, is considered to be an object with a unique identifier, the identifiers of the source and destination hosts in the topology, and a payload size in bytes.

The network entry point is tasked with receiving an incoming message, adding the current simulation time as metadata to the message, and forwarding it to the source host in the network model. The network model mimics the related delay and sends the message to the network exit point, which adds the current simulation time as metadata to the message object. The network analyzer is the core component that orchestrates the entire process. It reads in a file that describes the network traffic, sends this traffic to the entry point accordingly, receives the messages from the exit point, and writes the resulting messages with their modeled delays to a JSON file.

GeNSim, the custom data framework and CLI described in Section 5 is responsible for providing the network topology and traffic data to the OMNeT++ simulation. Whether we are testing a real-life topology or a generated one, GeNSim is able to convert a JSON file describing a topology into .ned and .ini files that can be directly imported into OMNeT++. Furthermore, the network analyzer was built to read files in GeNSim JSON format directly, removing the need for a conversion step. As long as the data is in the expected JSON format, it does not matter if it is real or synthetic.

This simulation framework is designed to be network model agnostic, allowing us to treat the model as a "black box"; network messages go in, and after a certain delay, they come out. This modular design simplifies the evaluation process, as illustrated by Figure 7.1, which shows how it effortlessly connects to both the single-node Latency-Rate model (left) and the multi-node INET model (right). Furthermore, separating this logic into three distinct components ensures that the network analyzer logic can be removed if needed. This enables the black box framework to not only be used in this experimental environment with traffic supplied by GeNSim, but also as a submodule of a larger simulation of the entire dCPS.

7.1.2 Experimental Environment

The experiment was conducted on a laptop running Windows 10 Education, Version 22H2, Build 19045.4412, an Intel Core i5-7200U CPU, and 8.00 GB RAM. Simulations were executed using OMNeT++ version 6.0.1, in the Cmdenv environment, utilizing 2 CPUs with 1 run per process. System consistency was maintained across all benchmark runs, meaning that, for example, no background processes were intentionally booted or closed, and the power cable was kept in at all times. All experiments were performed in randomized

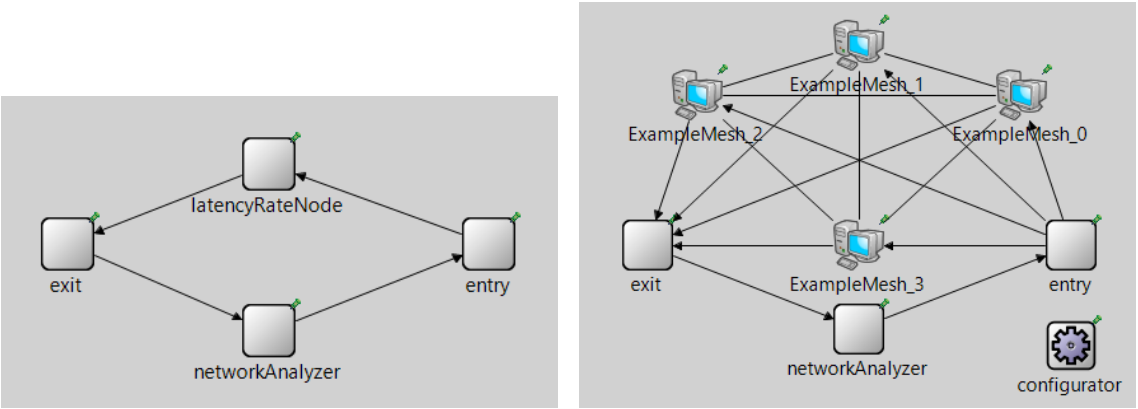

Figure 7.1: Two examples of the simulation framework, showing how a Latency Rate model (left) and an INET model (right) are instrumented using the entry point, exit point, and network analyzer.

order to mitigate the statistical impact of any unaccounted for temporal changes in the system (e.g. temperature, independent background processes). All of this is done to ensure that any trend in the results is less likely to be attributed to the environment in which the experiments were run.

7.2 Network Model Validation: Industrial Case Study

The purpose of this experiment is to validate the accuracy of the proposed network delay models using real-world data from an industrial setting. More specifically, network traffic data was collected from an ASML TwinScan Test Bench machine, which is a dCPS that is set up to mimic a real ASML Lithography machine setup. This section explains the steps required to prepare for and execute the model validation experiment, followed by a showcase of the results obtained from the experiment.

7.2.1 Data Collection and Preparation

For the model validation experiment, the ASML TwinScan network topology that the experiment was conducted on needs to be described in terms of our network model framework. This process can be executed manually, or a network may be crawled automatically using a tool such as OpenAudIT (20). In our case, internal documentation of the network was converted into the GeNSim topology representation using a simple Python script. This resulted in an abstract view of the network topology, as shown in Figure 7.2, which shows that the network that this case study is based on consists of one network switch that all

hosts are connected to, a main host, the brains of the operation, a data host, which handles all data traffic, and the embedded hosts, displayed at the top of the image, which handle the various specific tasks that the dCPS is built to carry out. The network shown in the case study is a subnet of a more complex network architecture.

Following that, the network traffic data was collected using a proprietary tool called the TwinScan Integrated Platform Performance Suite (TiPPS) (21). TiPPS is specifically developed by and for ASML to gather performance metrics of TwinScan machines, such as network message size, message sending time and message receival time. Since TiPPS operates on the software level, not on the network interface level, network message send and receive times are measured from the start of sending the message to the end of receiving the message. The delay is then calculated by subtracting the messages receival time from the message send time. Although TiPPS itself is proprietary, it served as the basis for a more broadly applicable, now open-source tool called the Platform Performance Suite (PPS) (22), allowing other researchers to collect and analyze similar data in their work. The relevant data for this experiment was extracted from the TiPPS models and converted into the GeNSim traffic representation. However, data collection was limited to only two of the dCPS subsystems, one being the main host of the TwinScan test bench machine, the other being one of the embedded hosts (marked by the red squares in Figure 7.2. This is because TiPPS is currently limited to only collecting traces from these types of machines.

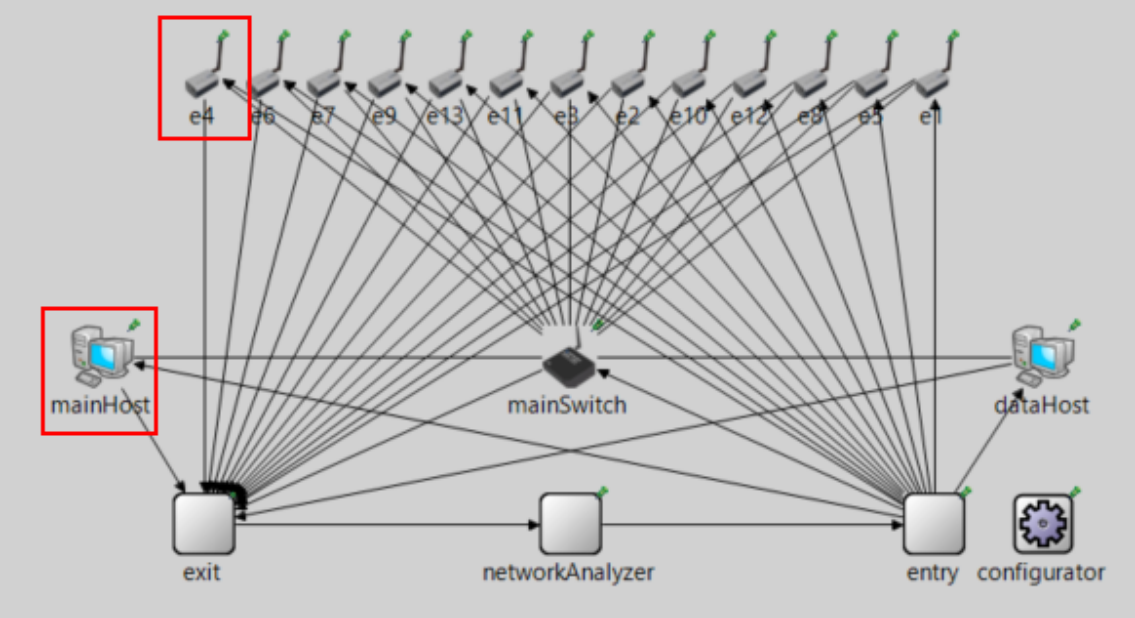

Figure 7.2: A schematic view of the network topology of the dCPS that was used for this case study. The observed hosts are marked in red.

To ensure the validity of the experiment data, a preliminary statistical analysis was performed. This analysis is essential for understanding the characteristics of the dataset and identifying any potential difficulties that could impact the results. The "Initial" columns of Table 7.1 show this statistical analysis for the two data sets which were collected with the purpose of parameter tuning and validating the models. The table includes the total number of messages in the network trace, as well as the minimum, maximum, mean, median and mode message delay in nanoseconds and message size in bytes.

An immediate observation one could make about this data is its skewness due to the apparent presence of outliers. The maximum message delay is in the order of seconds, yet the mean and mode are much closer to the minimum message delay of 500 nanoseconds. Moreover, Figure 7.3 shows that the bandwidth distribution, given by $\frac{size}{delay}$, is also skewed, with approximately 10% of the total data set having a relatively very low delay compared to other messages of similar size. This shows that bandwidth is inconsistent in the data set, thus there are probably also messages with relatively high delay compared to other messages of similar size. These inconsistencies also hint to the data set having faulty measurements.

Technically, these outliers could be a characteristic of actual network traffic. Though, even if that were the case, it is difficult to measure a model's performance with such skewed data, because extreme values can disproportionately affect the error metrics. After careful consideration and discussions with domain experts about this, quote, "fishy" data, outlier removal was suggested to obtain a more representative dataset and ensure that the models are evaluated on data that reflects typical network conditions.

To filter out the potentially anomalous data points, we employed the widely-recognized Interquartile Range (IQR) method for identifying outliers (23). This method calculates the quartiles of a sorted list of delays from all messages ($Q1$ and $Q3$) and uses those to determine the interquartile range ($IQR = Q3 - Q1$). Outliers are then defined as all data points below $Q1 - 1.5 \times IQR$ or above $Q3 + 1.5 \times IQR$. The messages are filtered to retain only those whose delay falls within the calculated acceptable range.

After applying the IQR outlier detection, the refined data set metrics are more evenly distributed and representative of typical network traffic. The "Filtered" columns of Table 7.1 show that, while only removing some 20% of the data, it was possible to get the max message delay and size much closer to the mean and median. Besides that, the mean and median message delays and sizes were minimally affected, further illustrating the initial dataset's skewness. Furthermore, Figure 7.4 shows a post-outlier-removal histogram which looks very similar in shape to the pre-outlier-removal histogram, other than that

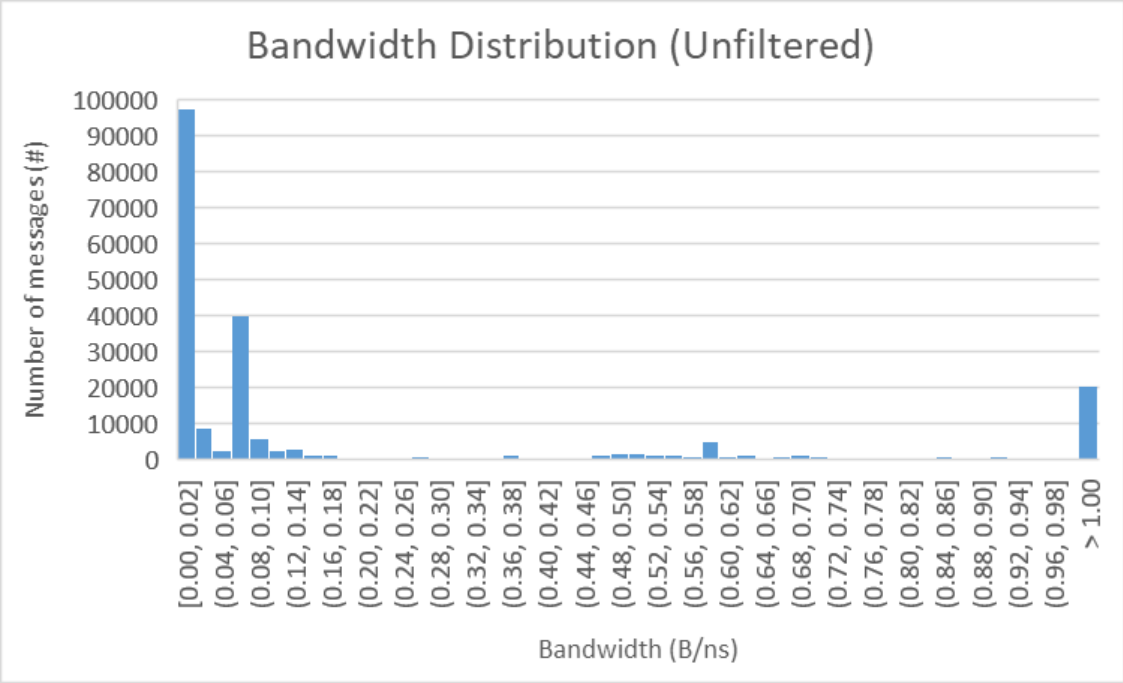

Figure 7.3: Bandwidth distribution of the initial train data set

the extreme buckets $[0, 0.02]$ and > 1 have drastically shrunk in size or have been removed entirely.

Another observation of the initial data set is that nearly 45% of messages have the exact same delay of 500 nanoseconds. This suggests the presence of measuring errors in our data set, since it seems highly unlikely that this many messages would have the exact same delay on nanosecond scale. This is likely due to a lower bound in TiPPS measurements. Unfortunately, there is no way to mitigate this issue. However, the presence of many messages with similar delays, especially with smaller sizes, is not at all unrealistic.

7.2.2 Model Parameter Tuning

All of the proposed models, except INET, have parameters that affect their calculations, as discussed in Section 6. Using random parameter values would yield just as random and thus inaccurate results. Parameter tuning needs to be employed in order to retrieve meaningful results from the models.

First off, as was briefly mentioned before, two data sets are extracted from the complete network trace, a training set and a testing set, to prevent overfitting. Parameters are optimized using the training set by comparing the predicted delays against observed delays from the actual traffic, optimizing for the Mean Squared Error, given by $MSE = \frac{\sum(y_i - p_i)^2}{n}$.

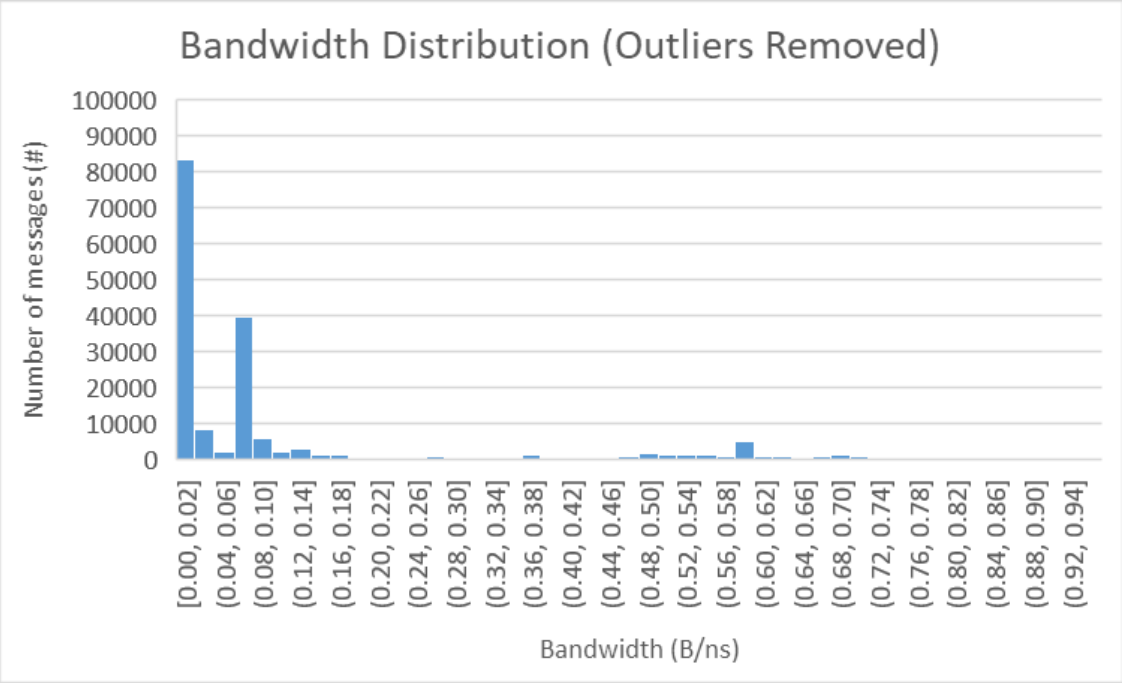

Figure 7.4: Bandwidth distribution of the train data set after filtering for outliers

MSE is chosen because, due to the squaring of the error term, the error becomes absolute and it penalizes large errors more heavily.

For the single-parameter models, the optimal parameter can be found using a simple brute-force optimization, in which the parameter is iteratively adjusted until the parameter value that yields the lowest MSE is found. In the case of the Latency-Rate model, calibration is a bit less straightforward. Since the best a simple statistical model can do to optimize for MSE is return the mean delay, the constant delay model will always yield the best training MSE. As the rate approaches infinity, the latency-rate model effectively converges to the constant delay model, nullifying the benefit of having two parameters. To mitigate this issue, the rate parameter is fixed to the optimized parameter value from the Constant Bandwidth model. The latency parameter is then optimized using a brute force method, similarly to the single parameter models. This approach simplifies the optimization process and retains the power of the Latency-Rate servers' dual parameter setup, while avoiding an excessively large optimization space.

The results of the parameter tuning phase are presented in Table 7.2. The table shows, for each model, the parameter values that achieved the best MSE during training. These optimal parameters were then applied to the test set. The table compares the MSE obtained on the test set with the lowest possible MSE that could be achieved by each model on

Metric	Train Data		Test Data	
	Initial	**Filtered**	**Initial**	**Filtered**
Count	204,411	164,736	261,376	207,948
Message delay (ns)				
Min	500	500	500	500
Max	1,787,313,683	45,701	2,945,634,317	39,695
Mean	52,273	9,139	106,358	7,898
Median	5,775	5,451	4,949	4,718
Mode	500 (43.44%)	500 (43.15%)	500 (43.68%)	500 (43.61%)
Message size (B)				
Min	32	32	32	32
Max	2,893,456	472	2,893,456	472
Mean	2,072	89	2,042	91
Median	40	40	40	40
Mode	40 (20.37%)	36 (22.85%)	40 (23.45%)	32 (22.21%)

Table 7.1: Comparison of metrics between initial data set and data set with outliers removed for both the train and test data sets. Message delay is reported in nanoseconds, message size is reported in bytes. The mode also reports the percentage of messages that have that value.

that same test data set. This difference in MSE highlights how well each model generalizes beyond the training data.

Model	Train MSE	Param	Test MSE	Best Test MSE	Difference
CD	1.131	9139 ns	0.911	0.894	0.0168
CB	1.659	0.023 B/ns	1.313	1.302	0.0108
LR	1.376	5183 ns	1.084	1.080	0.0033
INET	1.689	N/A	1.282	N/A	N/A

Table 7.2: Parameter tuning results of the Constant Delay (CD), Constant Bandwidth (CB), Latency-Rate (LR) and INET models. MSE (in s^2) is divided by 10000 for legibility.

7.2.3 Results

After the network traffic data is collected and prepared, and the model parameters are tuned, the real-world traffic data is used as input for the simulations, plugging each of the four proposed network models into the simulation framework. For each experiment run, the network analyzer creates a new file with the simulation time that each message entered and exited the network. This data is used to compare the output of the network delay

models with the real-life traffic using the appropriate error metrics. Table 7.3 provides a complete set of accuracy results obtained from the experiment reported in nanoseconds, divided by 10,000 for legibility. In this context, the "error" is defined as the predicted delay minus the actual delay. This means that a positive error indicates the model overshooting the delay and a negative error means that the model predicted the delay to be less than it actually was.

Model	MSE (ns^2)	Mean Error	Median Error	Min Error	Max Error
CD	1.131	0.0000462	0.369	-3.656	0.864
CB	1.659	-0.5260587	-0.196	-4.431	2.002
LR	1.376	0.0000768	0.380	-3.913	3.709
INET	1.689	0.7134636	0.357	-8.739	4.475

Table 7.3: Error metrics for the Constant Delay (CD), Constant Bandwidth (CB), Latency-Rate (LR) and INET models. All values are reported in nanoseconds, except Mean Squared Error, which is reported in nanoseconds-squared. Errors are divided by 10,000 for legibility.

Notably, the INET model, despite having no parameters that need to be tuned, performs very similarly to models with tunable parameters, especially on the mean squared error and median error metrics. Additionally, the Constant Delay and Latency-Rate models show very similar values across nearly all metrics.

7.3 Network Model Benchmarking: Synthetic Network Topologies and Traffic

This experiment aims to measure each model's simulation speed and scalability using synthetic network traffic and topology data generated by GeNSim, as discussed in Chapter 5. Since DSE of dCPS involves evaluating a design space with a potentially massive number of possible configurations, a model that is suitable for this use case must demonstrate this property.

7.3.1 Experimental Setup

The goal of this experiment is to test two factors of scalability: scalability of network topology complexity, and scalability of network traffic data size. These are two factors that are important for a model to handle in the setting of DSE for dCPS, because evaluating design points involves simulating traffic for an arbitrary large workload given an arbitrary large network topology. To assess the former scalability factor, we used NSIM to generate

fully-connected mesh topologies, each with an incrementally larger number of nodes, and traffic patterns for those topologies, each following a Poisson distribution with a lambda (the mean arrival rate per second) of 100 and a duration of 1000 seconds. This ensures that we test the network topology scalability against a stable, somewhat realistic traffic pattern. The latter scalability factor was evaluated by generating one mesh topology with 20 nodes and a connectivity of 0.5, followed by generating various Poisson traffic patterns with lambda = 100 and an incrementally larger duration, to test how well the models fare against larger and larger network traffic traces.

The experiment was then conducted by taking each network topology configuration and running it through the simulation, using the synthetic traffic as input. Each configuration was tested multiple times to account for variability. A common practice is to run each configuration at least 30 times to obtain a reliable measure of central tendency. Furthermore, all models were tested with the optimal parameters determined in the previous experiment, since model accuracy is not measured during this experiment and the parameters are not expected to have a significant effect on the simulation speed.

7.3.2 Results

Figures 7.5 and 7.7 show the results of both speed experiments the form of line charts. However, since the INET model seems to be significantly slower than the other models, Figures 7.6 and 7.8 present the same data, excluding the INET model to better see the nuances in the other models' speeds.

In the case of scalability of network topology complexity, all analytical models (Latency-Rate, Constant Bandwidth, and Constant Delay) seem to be unaffected by the increase in topology complexity. Among these models, the Constant Bandwidth model is marginally slower than the other two. However, this speed difference is negligible compared to the substantial speed difference with the INET model, which is orders of magnitude slower than the other models. Besides that, the INET model slows down significantly in simulation speed as the topology increases in complexity. This is very much expected, since the INET model is the only one that simulates all network elements in the topology. Growing the topology increases the complexity of the INET model and the number of hops it has to simulate for a message to be sent from one host to another. This does not count for the other models, which rely solely on simple arithmetic operations to approximate the delay of an entire topology at once.

The scalability of network traffic test reveals similar findings: The INET model is significantly slower than the other models. The Constant Bandwidth model is again slightly

slower than the Latency-Rate and Constant Delay models. However, all analytical models evaluated the same number of discrete events for each test, which is expected since they all do one calculation per arrival. In this test all models exhibit linear growth in execution time relative to the duration of network traffic, indicating predictable scaling behavior. This is also expected, since messages in the simulation models are independent. One message entering the system should generate approximately ten times less events than a trace of ten events in the network, which is now confirmed by this experiment.

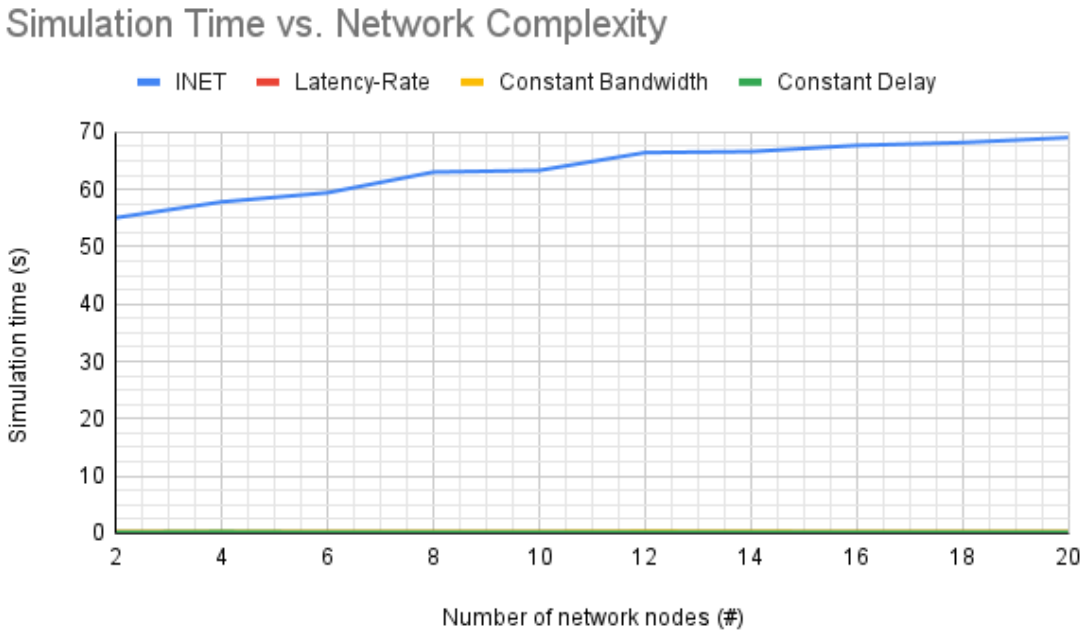

Figure 7.5: The model simulation time over network complexity.

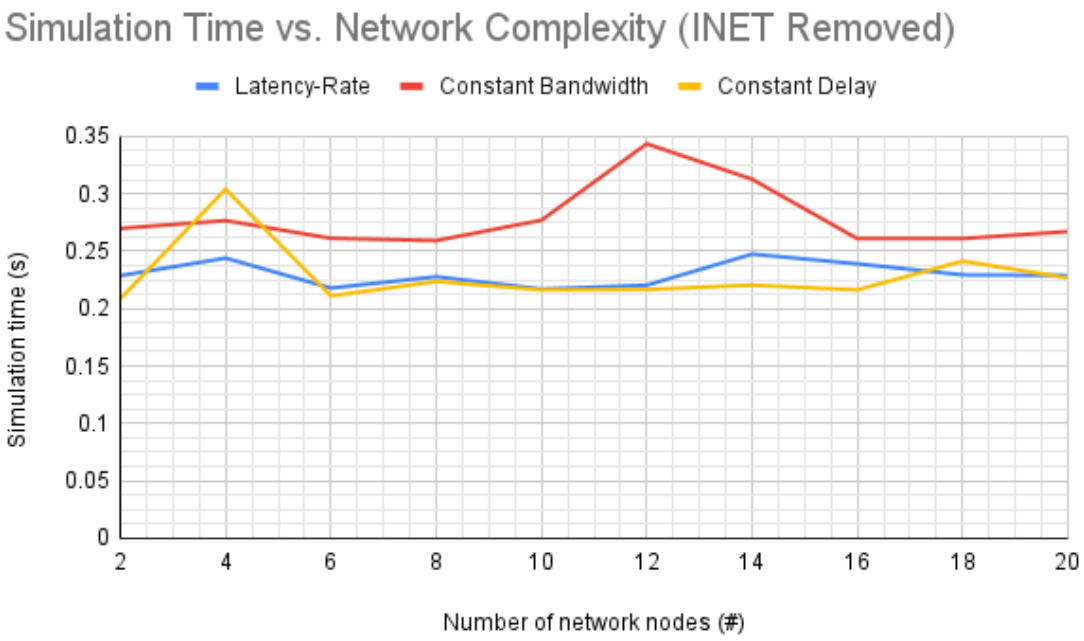

Figure 7.6: The model simulation time over network complexity without the INET model.

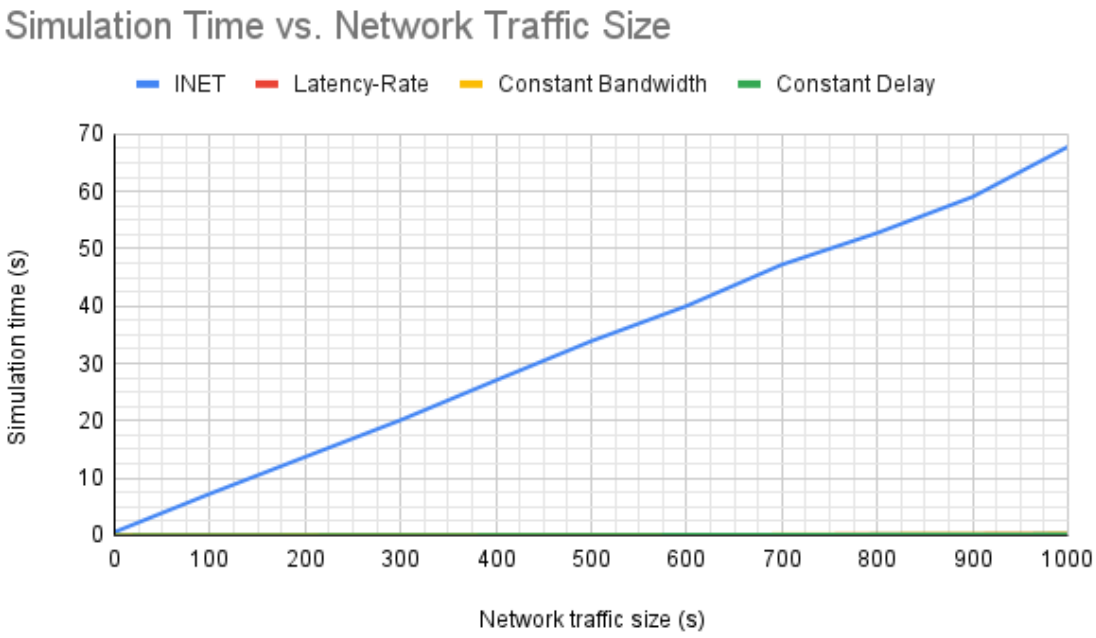

Figure 7.7: The model simulation time over network traffic duration.

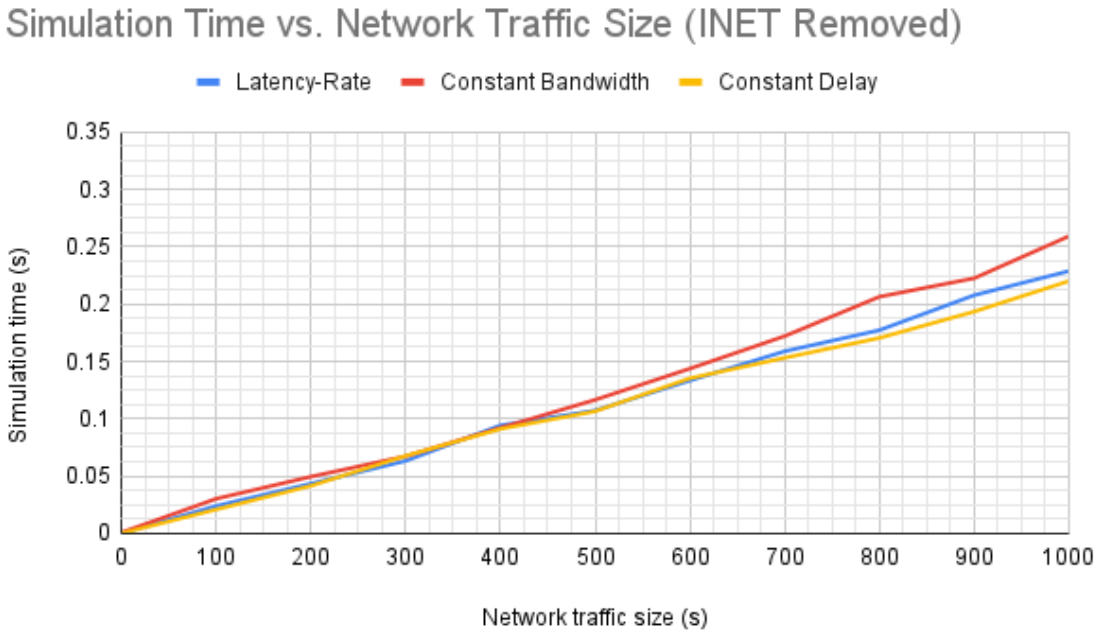

Figure 7.8: The model simulation time over network traffic duration without the INET model.

8

Discussion

This section compares the findings from Chapter 7 regarding the models defined in Chapter 6 along the three key measures for DSE applicability in dCPS as outlined in Chapter 4: speed, accuracy, and usefulness.

Speed is a straightforward measure to discuss, since the INET model is orders of magnitude slower than the Constant Delay, Constant Bandwidth, and Latency-Rate models. Among the analytical models, the Constant Bandwidth model is slightly slower than the Constant Delay and Latency-Rate models, though this difference arises from the arithmetic operations required to evaluate them, rather than the number of events generated by the model, making the speed difference negligible. Furthermore, the INET model is unique in simulating each host in the topology individually, unlike the analytical models that model the entire topology with a single calculation. This means that the INET model becomes slower as the topology becomes more complex, having to simulate an increasing number of network elements. Depending on the number of design points and the size of the network traces that need to be evaluated, the INET model's scalability issues and its relative slowness may render it less suitable for DSE of dCPS than the analytical models. Conversely, the other models, owing to their high level of abstraction, are much faster and hardware resource-friendly, which comes in handy in the case of extensive design space explorations.

In terms of accuracy, the Constant Delay model outperforms the other models discussed in this Thesis in nearly all error metrics. This result is expected, since the Constant Delay model is designed to return the mean delay, naturally optimizing for the Mean Squared Error metric given a network traffic trace. It is, however, interesting to note that the other models, especially the Latency-Rate model, are often not far behind the Constant Delay model. All models are able to achieve a Mean Squared Error in the order of 10,000 ns^2 on both data sets, which, depending on the use case may be accurate enough for DSE

of dCPS. However, the difference between the MSE a model has achieved on the test set and the best MSE that model could have gotten in the test set, shown in the "Difference" column in Table 7.2, paints a different picture of how sensitive the models are to unseen data. While the Constant Delay model boasts strong accuracy metrics, it overfits massively to the training dataset, resulting in poor generalization. This is inherent to the model, it is very sensitive to being trained on a different data set because it takes on the shape of the data set. The Constant Bandwidth model generalizes slightly better than the Constant Delay model by taking message size into account. However, the Latency-Rate model shows the best generalization by far due to its dual parameter setup, which prevents overfitting more effectively than the single-parameter models.

Finally, the usefulness of a model is assessed by its ability to handle the core DSE activities of: (1) altering network bandwidth, (2) altering network latency, (3) altering both bandwidth and latency simultaneously, or (4) altering neither. The Constant Bandwidth model works best in cases altering only the bandwidth, while the Constant Delay model works best for cases in which latency is altered. The Latency-Rate model, due to its dual parameters, can handle changes in both bandwidth and latency separately or simultaneously. Case 4 considers DSE without altering the network at all. This might be true when optimizing the workload or computational resources of the dCPS, using the network model within a greater dCPS model. All models discussed in this thesis satisfy this use case.

Given these considerations, one might conclude that the Latency-Rate model is the most applicable for DSE of dCPS. Out of the four evaluated models it is one of the fastest, one of the most accurate, and one of the most generally useful. It appears to be right in the sweet spot of suitability. However, even though the statistical models like Constant Delay, Constant Bandwidth, and Latency-Rate perform well with tuned parameters, they become ineffective without traces to optimize against. Without tuning their parameters, the calculations they perform are completely random and meaningless. The INET model is the only model evaluated in this study that directly incorporates topology information into its calculations, simulating the topology node-by-node without needing a parameter tuning phase. This is evident as the INET model achieved comparable MSE scores to the tuned statistical models. Tuning statistical models based solely on network topology properties is challenging, reinforcing the INET model's utility for DSE of dCPS.

It seems that no single model satisfies all measures of applicability confidently. Therefore, this thesis proposes a dual-step modeling approach, in which different models are leveraged for their respective strengths. In our evaluation, the INET model matches in accuracy metrics with the analytical models, without the need for parameter tuning. Therefore, one

could assume that the INET model, despite its slowness, could generate delay traces for unknown network topologies that are somewhat accurate to what real network traffic would look like in that topology. The INET model could create a trace based on a sample of the total data, which can then be used to tune the Latency-Rate model, which, although fast and accurate post-tuning, is otherwise inaccurate and thus not useful. This tuned Latency-Rate model is then used for the full simulation. The Latency-Rate model was chosen for this approach over the Constant Delay and Constant Bandwidth models, because it achieves similar accuracy and speed metrics as these models, but achieves all four use-cases of usefulness for DSE of dCPS, which the others don't. This dual-step approach of calibrating one model based on data generated by another model harnesses both models' strengths and mitigates their weaknesses. Future work should explore the feasibility and effectiveness of the opportunity that is this combined approach.

That being said, a major limitation of this thesis were the challenges posed in the collection of real-world dCPS networking data. We were only able to collect data of one case study. This severely limits the scope of the accuracy experiment. A large evaluation with more real-world networking traffic would allow for examining in which (edge) cases one model may be more accurate than another. Furthermore, of this single case study, data collection was limited to just two out of many hosts available in the modeled subnet, as shown in Figure 7.2. This means that we were not able to capture all interfering traffic in the main switch, only the traffic between the two hosts that allowed us to run our tracing software on. For the INET framework to be accurate, it needs detailed information about all other traffic flows in the system as well. Otherwise, it cannot compute the contention. If this information is not available, like in our experiment, then the contention needs to be limited in the system for the model to be accurate. Therefore, one could state that the INET model in our experiment was accurate by accident, and that, given a network trace with different network utilization characteristics, the INET framework could produce far less accurate results. Future work should investigate how the INET framework performs in network environments with differing utilization and congestion characteristics.

9

Conclusions and Future Work

This thesis investigated the creation and validation of network delay models for the use of Design Space Exploration (DSE) for Distributed Cyber-Physical Systems (dCPS). The primary objective was to compare network delay models to strike a balance between accuracy compared to real-world systems, simulation speed, and general usefulness for DSE of dCPS. The research contributions are threefold: the formalization of network topology and traffic concepts, a proposal of four network delay models at different levels of abstraction, and the comprehensive experimental evaluation of these models.

9.1 Summary of Contributions

First, the **formalization of network topology and traffic concepts** laid the groundwork for representing and generating networking data by defining their key components. We developed an open-source data framework and CLI application called *GeNSim* to represent and generate networking data systematically. GeNSim provides standardized representations for network topologies and traffic patterns, supporting both real-world and synthetic data. It is also able to convert data between the supported representations, validating the data to confirm it is formatted according to the specification, and generating synthetic network topologies and traffic data that can be used to test the proposed models.

Second, leveraging the established framework, **four network delay models were proposed**, each operating at a different level of abstraction. The *Constant Delay* model represents the most abstract form, providing a fixed delay for all messages, capturing the average network latency. The *Constant Bandwidth* model improves on this by considering the size of the messages, as well as a bandwidth parameter in calculating the message delay. The *Latency-Rate* model introduces a dual-parameter system based on the Latency-Rate

server abstraction, potentially offering greater flexibility to model the target system with greater accuracy. Finally, the *INET* model simulates the entire network in detail using the INET framework within OMNeT++, considering each node and interaction individually with high fidelity.

Third, the **experimental evaluation** involved a twofold approach: validating the models' accuracy using real-world data from an industrial case study, and assessing the simulation speed and scalability using synthetic data. The *accuracy assessment* was conducted using networking data from an ASML TwinScan Test Bench machine. We tuned the parameters of the analytical models using a training dataset and validated their performance on a separate testing dataset. The *speed assessment* was conducted by employing synthetic network topology and traffic data generated using GeNSim. We analyzed how the models scale with increasing network complexity and traffic volume.

The results of these evaluations revealed some key insights. The Constant Delay and Latency-Rate models achieved the best accuracy in terms of mean squared error (MSE) when their parameters were tuned using real-world data. The Latency-Rate Model, with its dual parameters, offered better generalization on unseen data compared to single-parameter models. However, without parameter tuning, all analytical models' accuracy significantly diminishes, limiting their practical applicability in DSE where prior data may not be available. The INET model showed comparable accuracy to the tuned analytical models without the need for a parameter tuning phase.

In terms of simulation speed, the analytical models significantly outperformed the INET Model. The analytical models demonstrated linear scalability with respect to traffic volume and were unaffected by increases in network complexity due to their high level of abstraction. Conversely, the INET Model exhibited much longer simulation times and poorer scalability due to the computational overhead of its detailed simulation of network components. This may render the INET model impractical for DSE of dCPS, where numerous design points must be evaluated efficiently.

The network delay models were evaluated based on their ability to support the core DSE activities of altering network bandwidth, network latency, both, or none. The INET model directly simulates the network topology, making it the most useful of all models with respect to the aforementioned DSE activities. The Latency-Rate model was the most versatile analytical model, capable of independently adjusting latency and rate parameters to reflect changes in the network. The Constant Delay and Constant Bandwidth models were limited to single-parameter adjustments.

9.2 Conclusions

Given these findings, this thesis concludes that no single model fully satisfies all three key measures of accuracy, speed, and usefulness for DSE of dCPS simultaneously, demonstrating the presence of a tradeoff. The INET model, while accurate without parameter tuning, is computationally intensive and scales poorly, making it unsuitable for rapid exploration of large design spaces. The analytical models are computationally efficient and, when tuned, can achieve relatively good accuracy. However, their reliance on parameter tuning based on specific datasets limits their applicability in scenarios where prior data is unavailable.

To address these limitations, a novel dual-step modeling approach was proposed, which leverages the INET model to generate synthetic network delay traces on an unseen topology, which are then used to tune the Latency-Rate model for efficient and accurate full simulation runs. This approach leverages the strengths of both models and mitigates their weaknesses.

9.3 Future Work

Future work should further refine the proposed dual-step modeling approach, exploring its practical application in real-world DSE scenarios. Besides that, additional research is needed to validate the effectiveness of the dual-step modeling approach in practical DSE settings:

- **Extension to diverse network configurations:** Future studies should evaluate the models across a wider range of network topologies and traffic patterns, including those with varying levels of congestion and variability. This would provide a more comprehensive understanding of the models' accuracy and generalization capabilities.
- **Automated parameter tuning:** Developing automated methods for parameter tuning of analytical models, possibly using machine learning techniques, could enhance their applicability in DSE without relying on prior data.
- **Integration with a workload model:** Integrating the proposed models into a dCPS workload model would facilitate their adoption in industrial DSE settings.

References

[1] Marius Herget, Faezeh Sadat Saadatmand, Martin Bor, Ignacio Gonzalez Alonso, Todor Stefanov, Benny Akesson, and Andy D. Pimentel. **Design Space Exploration for Distributed Cyber-Physical Systems: State-of-the-art, Challenges, and Directions**. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 632–640, Maspalomas, Spain, August 2022. IEEE. iii, 1, 2, 5, 6, 14

[2] Manil Dev Gomony, Jamie Garside, Benny Akesson, Neil Audsley, and Kees Goossens. **A Globally Arbitrated Memory Tree for Mixed-Time-Criticality Systems**. *IEEE Transactions on Computers*, **66**(2):212–225, February 2017. iii, 8, 9

[3] Andy D. Pimentel. **Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration**. *IEEE Design & Test*, **34**(1):77–90, February 2017. 6

[4] Minyoung Kim, Sudarshan Banerjee, Nikil Dutt, and Nalini Venkatasubramanian. **Design Space Exploration of Real-Time Multi-Media MPSoCs with Heterogeneous Scheduling Policies**. In *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, pages 16–21, Seoul Korea, October 2006. ACM. 6

[5] Francisco J. Suárez, Pelayo Nuño, Juan C. Granda, and Daniel F. García. **Computer Networks Performance Modeling and Simulation**. In *Modeling and Simulation of Computer Networks and Systems*, pages 187–223. Elsevier, 2015. 8, 9, 11

[6] Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall/CRC Texts in Statistical Science Series. CRC Press, Taylor & Francis Group, Boca Raton, FL, third edition edition, 2017. 8

REFERENCES

[7] Leonard Kleinrock. *Queueing Systems*. Wiley, New York, 1975. 8

[8] T. Murata. **Petri Nets: Properties, Analysis and Applications**. *Proceedings of the IEEE*, **77**(4):541–580, April 1989. 8

[9] D. Stiliadis and A. Varma. **Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms**. *IEEE/ACM Transactions on Networking*, **6**(5):611–624, Oct./1998. 8

[10] Mohsen Guizani, editor. *Network Modeling and Simulation: A Practical Perspective*. Wiley, Chichester, West Sussex, U.K, 2010. 9

[11] Andras Varga. **OMNeT++**. In Klaus Wehrle, Mesut Güneş, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 35–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. 9

[12] **NS-3 - a discrete-event network simulator for internet systems**. `https://www.nsnam.org/`. Accessed: 2023-05-09. 9

[13] Herman Kelder. **Exploring Scalability in System-Level Simulation Environments for Distributed Cyber-Physical Systems**. 2023. 10

[14] Robert G. Sargent. **Simulation Model Verification and Validation**. 1991. 10, 11

[15] Rajive Bagrodia and Mineo Takai. **Position Paper on Validation of Network Simulation Models**. *Computer Science Department of University of California, Los Angeles*, 1999. 10

[16] Thomas H. Naylor and J. M. Finger. **Verification of Computer Simulation Models**. *Management Science*, **14**(2):B–92–B–101, October 1967. 10

[17] Svilen Ivanov, Andre Herms, and Georg Lukas. **Experimental Validation of the Ns-2 Wireless Model Using Simulation, Emulation, and Real Network**. January 2007. 10

[18] Hardik Shah, Alois Knoll, and Benny Akesson. **Bounding SDRAM Interference: Detailed Analysis vs. Latency-Rate Analysis**. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pages 308–313, Grenoble, France, 2013. IEEE Conference Publications. 13

[19] Andrew Nelson, Kees Goossens, and Benny Akesson. **Dataflow Formalisation of Real-Time Streaming Applications on a Composable and Predictable Multi-Processor SOC**. *Journal of Systems Architecture*, **61**(9):435–448, October 2015. 14

[20] **Introducing Open-AudIT**. 25, 29

[21] Nieke Roos. **Taking performance analysis to the system level**. `https://bits-chips.nl/article/taking-performance-analysis-to-the-system-level/`, August 2023. 30

[22] Kostas Triantafyllidis. **PPS - Platform Performance Suite**, June 2024. 30

[23] Peter J. Rousseeuw and Christophe Croux. **Alternatives to the Median Absolute Deviation**. *Journal of the American Statistical Association*, **88**(424):1273–1283, December 1993. 31