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

William Ford
Department of Computer Science
University of Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands
w.a.ford@student.vu.nl

***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 prototypes has grown in complexity beyond human comprehension. New research in the field of scalable automated Design Space Exploration (DSE) is necessary to engage with this challenge. A model of the dCPS must be created to enable DSE, with one crucial component being the network that connects the dCPS subsystems. The network delay needs to be systematically evaluated in order to effectively explore the design space. Previous work in the fields of analytical network modeling, network simulation, and network model validation is reviewed. In addition, a recommended plan is presented to create and validate such a network model based on this previous work.**

***Index Terms*—Design space exploration, distributed cyber-physical systems, discrete event simulation, network model validation**

May 30, 2023

1. Introduction

Distributed Cyber-Physical Systems (dCPS) are heterogeneous multi-core or many-core systems, comprising distributed subsystems that are connected via complex networks. Nowadays, dCPS are one of the largest innovation-driving forces behind industrial sectors such as: health industries, industrial automation, robotics, and avionics [6]. However, in recent years, dCPS have grown in complexity. Traditional methods of evaluating dCPS designs, such as building prototypes, have become too complex, time-consuming, and costly.

In order to engage with this challenge, new research in Design Space Exploration (DSE) methods for dCPS is required. DSE refers to the process of automatically searching the *design space* of all possible dCPS configurations for one or more configurations that best satisfy the design objectives. Given a set of design objectives, this process severely reduces the dCPS design costs.

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, achieving marginal reductions in execution time can yield large cumulative gains. More research in the context of design space pruning, simulation efficiency and scalability must be done to engage with this challenge. Second, in order to evaluate a design point, the simulation model needs to be able to capture both the phyisical hardware components and the software components of the system. In order to do this, one crucial part of the simulation model is the network model, which needs to be able to model the complex network interactions typically seen between dCPS processes.

DSE2.0 [6] 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 machines. The objective of this literature study is to develop a framework for a process, based on academic literature, that creates and validates a network delay model to be used in the context of DSE for dCPS.

The remainder of this literature study is structured as follows: Section 2 provides more depth and context to the concepts introduced in this section. Section 3 reviews literature related to the problem domain. Furthermore, Section 4 discusses the findings from the related work section and proposes a framework for creating and validating a network delay model to be used in the context of DSE for dCPS. Finally, Section 5 provides a conclusion on the literature study and its findings.

2. Background

The goal of the background section is to introduce the basic concepts required to understand this literature study, as well as inform the reader of the broader context in which this literature study was produced. First, the concept of Design Space Exploration (DSE) is introduced in Section 2.1. Then, the goal of this literature study, given the broader context of DSE, is discussed in Section 2.2.

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 [6]. The DSE process typically consists of four main stages (see Figure 1):

1) **Modelling**: the system is discovered, described, abstracted, and mapped to a system representation; 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 spot.
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) [17] and Multiprocessor System-on-a-Chip (MPSoC) [9]. However, the growing complexity of distributed Cyber-Physical Systems (dCPS) poses several challenges for DSE, such as combinatorial explosion and simulation scalability.

2.2. Network modeling

Network modeling involves the abstraction of network features and properties, enabling the creation of analytical representations at varying levels of complexity [22]. Various types of analytical representations are considered in Section 3.1. Such an analytical network model may be used in a computer simulation environment to evaluate network configurations, as is further discussed in Section 3.2. In addition, a network model may be used to generate synthetic traffic patterns that mimic the behavior of the system under different operating conditions. This may help researchers to identify performance bottlenecks or reliability issues with a given network configuration.

As mentioned in Section 2.1, one of the key challenges of applying DSE to complex dCPSs is the modelling of heterogeneous subsystems. In order to engage with this challenge, a network model must be created that is able to capture the network delay of interactions between the interconnected dCPS subsystems. Due to the large-scale nature of dCPS design spaces, it is important that the execution time of the evaluation of a design point is kept to a minimum.

The relationship between the accuracy and the speed of a computerized model is referred to as the *speed-accuracy tradeoff*. The accuracy of the model refers to how closely it can replicate the behavior of the real-world network, while speed refers to the speed at which a design point can be evaluated. Generally, as the level of accuracy increases, the computational complexity and execution time of the simulation also increase. Conversely, reducing the level of accuracy can lead to faster simulations, but at the cost of potentially inaccurate results.

In the process of model creation, it is essential to establish well-defined Key Performance Indicator (KPI) goals that effectively balance speed and accuracy. Some common examples of KPIs might include the Root Mean Squared Error (RMSE) for assessing accuracy, and simulation time for gauging speed. These KPIs must be measured for models

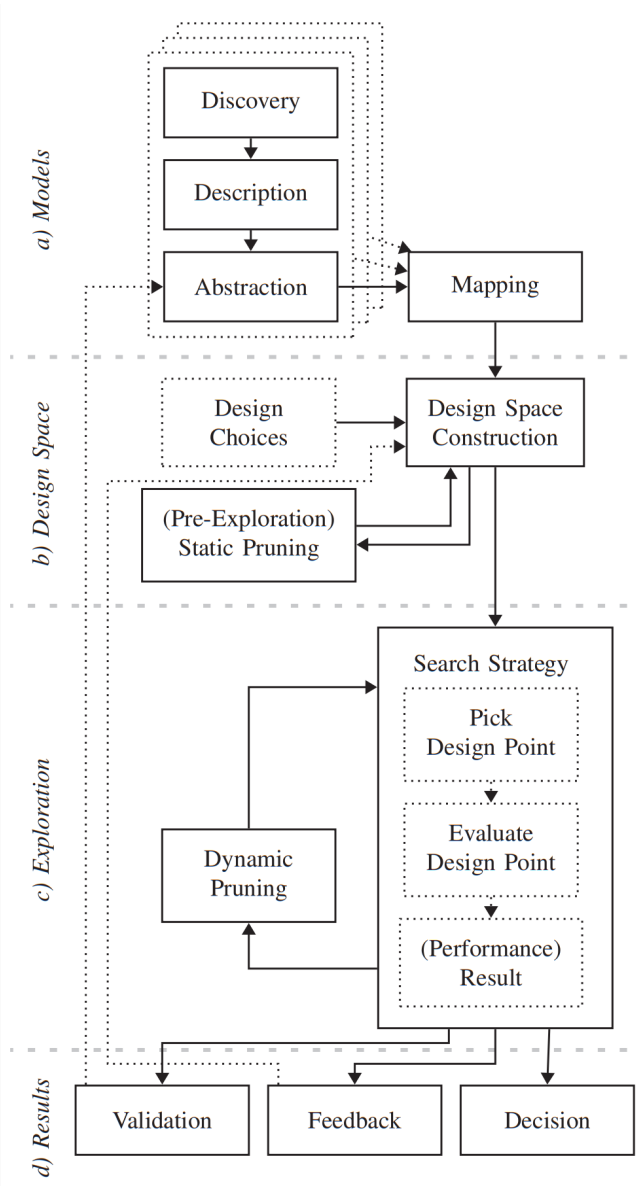

Figure 1. The general Design Space Exploration workflow [6].

that are created at varying levels of abstraction in order to select a model that is both fast and accurate enough to be used in a DSE context.

However, without verifying and validating a network model, DSE judgements made on its results may be misguided. Therefore, model verification and validation, as further discussed in Section 3.3, is a crucial step in the network model creation process. By incorporating a validated network model of suitable performance and abstraction into the DSE process, researchers can create a design space that spans all possible network configurations that fit the design objectives.

3. Related Work

This section provides a literature review of various concepts related to the problem domain presented in Section 1. First, Section 3.1 considers techniques for analytical network modeling, which is a modeling technique that obtains an analytical solution given some network input parameters. Section 3.2 discusses the topic of discrete-event simulation, which simulates network events to obtain network behaviour metrics. Finally, some approaches regarding validation of computer network models are discussed in Section 3.3.

3.1. Analytical Modeling

A system that evolves randomly in time, such as a computer network, can be modeled by a *stochastic process* [11]. 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 $X(t), t \geq 0$ [11].

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) [11]. 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 [22].

The remainder of this section will discuss various analytical modeling techniques that can be used in a stochastic context. These modeling techniques are broadly considered to be well-suited for the task of modeling computer networks. Each modeling technique will be discussed from the following viewpoints: definitions, the input and output of the model, and what the model can and cannot capture.

3.1.1. Markov Chains. Markov Chains are stochastic processes used to model the probability of transitioning between system states over time [11]. The system is assumed to have a finite state space and to occupy one of these states at any moment in time [20]. They are described using graph theory, where nodes correspond to system states and the directional edges represent the probability of transitioning from one state to another [22]. Markov Chains are low in analysis complexity, because they are memoryless; future states only depend on the current state and not on any previous states [22]. Figure 2 shows a simple example of a Markov Chain with two states. In this example, given the context of network modeling, state 0 could refer to the the idle state of a network, and state 1 could refer to the state where a packet has entered the network.

The input of a Markov Chain is the state space S, a transition probability matrix P, and an initial state X_0. The output is a probability distribution over the states in the chain, which represents the long-run, *stationary* probability of being in each state. The probability of being in a state at a particular time is called a *transient* probability [11].

Some of the metrics that a Markov Chain can capture in the context of modeling a computer network include: the probability of state transitions (e.g. a packet being transmitted from one node to another), steady-state behavior (long-term behavior of the network) and network performance in terms of factors such as delay, throughput, congestion, and packet loss [13]. However, many real-world networks show behavior that is not strictly Markovian, meaning that the probability of the network transitioning to a new state may depend on previous states. This makes it more difficult to accurately model directly with a Markov Chain [13].

Markov Chains assume stationary transition probabilities between states. However, the behavior of a network may change over time due to factors such as changes in traffic patterns or network topology. Furthermore, when the systems being modeled are complex, the number of possible states is high, and modeling them directly with Markov Chains becomes difficult. In these cases, more abstract models like Queuing Networks and Petri Nets may be better alternatives [22].

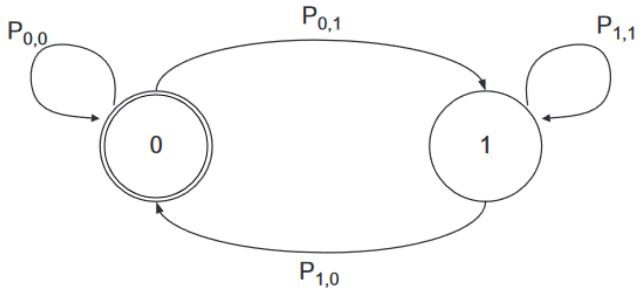

Figure 2. Example of a simple Markov Chain model with two nodes representing system states, and edges connecting the nodes with weights from the transition probability matrix P [22]. As indicated by the double stroke, state 0 is the initial state.

3.1.2. Queuing Networks. Queuing Networks are stochastic models used to analyze the behavior of systems that involve waiting in line, such as computer networks [10]. The model describes the flow of service requests through a system that contains service stations. When service requests arrive at a busy service station, they are queued according to a given queuing discipline until it is their turn to gain

access to the service station [22]. The behavior of the system is determined by the arrival rate of customers, the service rate of the servers, and the topology of the network. The key assumption is that both the arrival and service times of customers are described as stochastic processes that follow certain probability distributions [10]. Figure 3 shows a simple example of a Queuing Network model.

The input of a Queuing Network model consists of six components: the arrival pattern of customers A, the service pattern of servers B, the number of servers and service channels X, the system capacity Y, the queuing discipline Z (e.g. First-Come-First-Served (FCFS), Last-Come-First-Served (LCFS), or Random Selection for Service (RSS)), and the number of service stages. The output is a set of performance metrics, such as the average throughput, waiting time and queue length [19]. Different types of Queuing Networks are commonly described using Kendall-Lee notation: A/B/X/Y/Z. As an example: M/M/5/FCFS/20 may represent a bank with exponential arrival times, exponential service times, 5 tellers, an FCFS queuing discipline, and a total capacity of 20 customers [3]. It is common practice to omit capacity if no restriction is imposed and to omit the queue discipline if it is FCFS [19].

Some of the things a Queuing Network can capture in the context of modeling a computer network include: traffic flow (e.g. routing and processing of traffic through the network, identifying bottlenecks due to shared resources on distributed systems [22]), resource utilization (e.g. bandwidth, processing capacity, memory), and performance (e.g. response time, throughput, packet loss). An important advantage of simple Queuing Network models is the low complexity of the steady-state solutions (polynomial in number of queues and customers), because they can be obtained as a product of the steady-state solution for each of the individual queues in the network. This is possible because each queue operates independently of the others [22].

However, similarly to Markov Chains, some assumptions that Queuing Networks are based on may not hold in real-world networks. These include the assumption that arrival rate and service times follow some given stochastic process and the assumption that a network topology is fixed. Besides, Queuing Networks do not properly model the common synchronization mechanisms of distributed systems, something which Petri Nets are a better fit for [22].

3.1.3. Petri Nets. Petri Nets are a type of graphical and mathematical modeling tool used to describe the behavior of systems that involve concurrency, synchronization, and resource sharing [14]. They are described by a directed bipartite graph, with two types of nodes representing places / system states (circles) and transitions / system evolutions (bars). The edges are called arcs, which represent the flow of tokens (commonly associated to resources) between a place and a transition or vice versa [16]. The behavior of the system is determined by executing (or *firing*) the enabled transitions of the Petri Net. An enabled transition is a transition where all inputs have at least one token. Firing is an atomic operation that consumes one token from each input place and produces a token in its output places [22] [14]. Figure 4 shows an example of a simple Petri Net model before and after firing the transition.

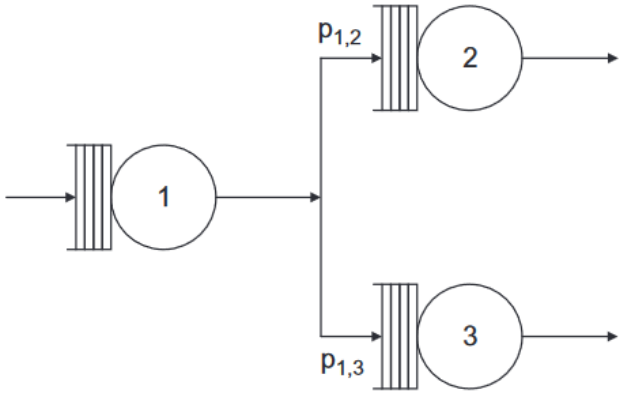

Figure 3. A simple example of a Queuing Network model with three service stations and the queues that service requests must move into before being served according to the given queuing discipline [22].

There are many types of Petri Nets, each of them enabling different analyses which make it possible to gain a deeper understanding of a given network [16].

- **Place-Transition Petri Nets**: What was described above. They can be used to model systems that do not have timing constraints.
- **Timed Petri Nets**: Similar to Place-Transition Petri Nets, but they include arcs with weights that represent the time needed to transition between states. This allows timed Petri Nets to capture the temporal behavior of a system, such as the response time.
- **Stochastic Petri Nets**: Petri Nets that include probabilities associated with their transitions, which represent the likelihood of an event occurring. They can be used to model systems that involve random elements.
- **Colored Petri Nets**: Petri Nets that allow for multiple 'colors' of tokens. They can be used to model complex systems with multiple types of data or traffic.

The input of a Petri Net model is a description of the system's structure: the places, transitions, and arcs, as well as a set of initial markings, which describe the number of tokens in each place in the system. The output of a Petri Net model is the set of reachable markings, which describes all possible states of the system that can be reached by executing the Petri Net [14].

Petri Nets are particularly useful for modeling concurrent systems, where multiple processes or events can occur simultaneously. From its output, a set of performance metrics may be deduced using reachability analysis, such as the number of tokens in each place, the throughput, and the response time [14]. Besides that, they can be used to model resource allocation (e.g. bandwidth, processing power), and system properties such as deadlock-freedom. However, the

number of possible states in Petri Nets scale exponentially with the network size. This means that Petri Nets may not be able to model large-scale computer networks. Besides, similarly to Markov Chains, and Queuing Networks, they are not able to handle topology changes in the network.

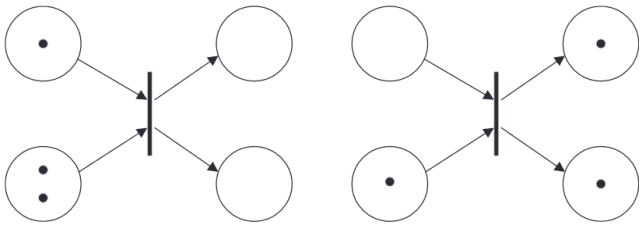

Figure 4. A simple example of a Petri Net model with four places, one transition and four arcs, two of them as input channels to the transition, two of them as output channels. Tokens are present at all the transition's input channels, which allows this transition to fire. The Petri Net is shown before firing (left) and after firing (right) [22].

3.1.4. Latency-Rate Servers. Latency-Rate Servers is an analytical model that is commonly used to analyze traffic scheduling algorithms. The model is characterized by two parameters: the latency and the service rate. 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 [21]. 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) [4]. A *system busy period* is a maximal interval of time during which the server is never idle [21]. Figure 5 shows an example of how a Latency-Rate server might react to incoming bursts of traffic [4].

The input of a Latency-Rate model is the maximum server latency Θ_i, and the minimum rate ρ_i. The output of the model is a set of performance metrics such as the maximum or mean response time [21]. Latency-Rate Servers differ from the other analytical models discussed above, because it is not necessarily a stochastic model. It may be used in a stochastic context if necessary, but the latency and rate parameters are not stochastic in nature, rather they can be tuned to achieve results that are similar to the behaviour of the real-life system.

The Latency-Rate server model can capture the interactions between packets in a distributed network, because the model takes the queuing behavior of packets into account as they travel through the network [21]. As opposed to Markov Chains, Queuing Networks and Petri Nets, the Latency-Rate server model is not fit to analyze a wide range of systems, rather it was specifically designed for modeling shared-resource systems. However, the model assumes that processing rates of the server are constant, which may not always be the case in real-world networks.

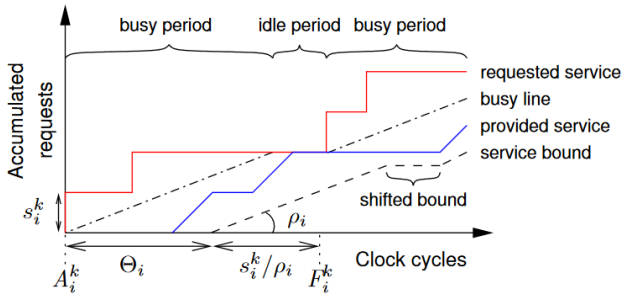

Figure 5. A Latency-Rate server and its associated concepts [4].

3.2. Discrete-Event Simulation

Discrete-event simulation 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 [5]. 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 [22].

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 [5].

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 several internet protocols and emulate network hardware [23]. Another example is NS-3, an open-source discrete-event network simulator targeted for research and educational use that supports a range of protocols and technologies [1].

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 [8]. Another challenge is the lack of standardized modeling approaches and simulation tools for dCPS. Table 1 in the appendix shows a high-level comparison of various state-of-the-art discrete-event simulation tools. 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.

3.3. Network Model Validation

Model validation is often defined as: "substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model". *Model verification* can be defined as: "ensuring that the computer program of the computerized model and its implementation is correct" [18].

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 verification and validation are 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. It is important to consider alternative methods and the requirements for a given project [15].

In the case of network simulation models, validation and verification may be done in the following contexts [2]:

- **Simulator verification**: Verifying that a simulation platform or tool is bug free.
- **Protocol validation and verification**: Ensuring that the simulation model of a network protocol faithfully replicates the real-world implementation of that protocol.
- **System validation and verification**: Ensuring that the simulation model of a physical resource faithfully replicates the real-world behaviour of that physical resource.
- **Scenario validation**: Proving that a reasonable relationship exists between the simulation experiments and the real life situations in which the corresponding technology will be deployed.
- **Methodology validation**: Validating that the experiment was designed with appropriate attention paid to removing statistical bias.

Since this literature study does not assume any given simulation platform or simulation model, for the remainder of this section, the term *validation* refers to the context of *scenario validation*. For the same reasons, the topic of *model verification* is out of scope for this section of this literature study.

One commonly accepted method to validate a simulation model is the Naylor and Finger validation approach [15], 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) [7]. 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) [18]. Turing tests by expert opinion are a popular way to incrementally validate a model [22].

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. Table 2 and Table 3 in the appendix show high-level overviews of several available network discovery and network measurement tools respectively to illustrate the diversity of choices. The network exploration tools listed in Table 2 in the appendix 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. The network measurement tools listed in Table 3 in the appendix 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.

4. Discussion

The discussion puts the literature that was reviewed in Section 3 in the context of the objective of this literature study: developing a framework for a process that creates and validates a network delay model given the context of DSE for dCPS. In Section 4.1 it will be discussed how a network model for DSE for dCPS may be created. Finally,

Section 4.2 discusses a validation approach for such a network model.

4.1. Network Modeling

From the literature, we can conclude that the analytical models on their own are not suitable for evaluation of large-scale dCPS networks due to their high complexity nature. However, the presented analytical models can be enhanced to model complex systems by using them in a computer simulation context. We will choose the OMNeT++ discrete-event simulation platform [23], because it is academically backed, it is open-source, and it includes several convenient libraries and frameworks that allow for the simulation of complex hardware, such as the INET framework [12]. Nevertheless, the findings of this literature study are expected to be applicable to any comparable discrete-event simulation platform (see: Table 1 in the appendix).

The objective of this literature study will be accomplished by developing network models at different levels of abstraction, and evaluating their speed-accuracy tradeoff to select a model for a given network configuration that is both fast and accurate enough to be used in the context of DSE for dCPS. The models that will be created fall into three different categories:

- **Functions**: Some fundamental, simple representations of any network. Some examples might be: return a constant delay, or return a delay that is correlated with the message size. The expectation for this model type is that they are relatively fast but inaccurate.
- **Analytical models in a simulation context**: Because of the limitations discussed in the related work section (mainly: complexity concerns and assumptions about network arrival patterns that do not hold in the real-world systems), we cannot rely on Markov Chains, Queuing Networks, Petri Nets to build a network delay simulation model of dCPS. However, the Latency-Rate Servers abstraction is relatively simple. It only has two parameters, thus it does not suffer from the complexity problems that the other analytical models have. Besides that, it is not stochastic in nature, which means it does not depend on some given arrival pattern. Furthermore, the requested service curve in Figure 5 can be generated from traces of actual network traffic, enabling researchers to evaluate the delay of that network traffic trace for different service latencies (Θ) and guaranteed service rates (ρ). The expectation for this model type is that they are moderately fast and moderately accurate.
- **Pure simulation models**: The discrete-event simulation platform will be used to create simulation models that are not necessarily based on any analytical models. These simulation models will range from "two nodes with a single link connecting them" to "the entire network topology closely emulated using the INET library", with some undefined abstraction steps in between. The expectation for this model type is that they are relatively slow but accurate.

These model types span the spectrum of model complexity, from the most simple, abstract, and fast model possible, to the most complex, representational, and slow model possible. It should be noted that this network model is only meant to capture the delay of messages between dCPS subsystems, not the actual functional behavior. This is why a very abstract model would suffice if it proves to be accurate enough. Even though we hypothesize that the network delay model becomes more accurate by adding more complex function elements to the simulation model, it is also possible that the delay model becomes less accurate if it is made with too much detail and complexity due to a heightened susceptibility to simulation errors. This is also why the model types are divided into these three categories, because it is interesting to include models in the speed-accuracy analysis that are both simpler and more complex than the analytical models discussed in the related work.

We will evaluate the created models using the following KPIs to measure the speed-accuracy tradeoff: the Root Mean Squared Error (per message transferred), the Total Error (all messages), and time to evaluate a design point. The former two KPIs are found by comparing the trace-driven simulation results to the actual traces of the real system. The latter KPI is calculated by evaluating a large number (i.e. 1000) of design points and taking the mean execution time.

4.2. Network Model Validation

Continuously during the model creation process, validation will play an active part of the process. We will largely follow the validation approach outlined by Robert G. Sargent (1991) [18]:

1) An agreement should be made between the model developing party and the model using party on the basic validation approach prior to developing the model.
2) In each model iteration, at least a face validity test should be performed. This step is used to determine if the logic of the conceptual model is correct, as well as validating the input-output relationships, covering all three bases of the Naylor and Finger validation approach [15].
3) In at least the last model iteration, comparisons should be made between the model and system behavior data for at least two sets of experimental conditions.
4) The validation approach and results should be described in the model documentation.

Within the process of validation, we are limited by the real-world data we are able to capture. Network topology data will be acquired using Open-AudIT, because it is open-source, works across platforms, is well documented, and it is possible to extend the tool with scripts to account for

possible edge cases in the dCPS network. For obtaining network measurements, Jaeger or Zipkin will be used. Both of these tools are open-source distributed tracing systems that are designed to visualize the timing and duration of requests in complex distributed systems. Similarly to the discrete-event simulation tool, the contributions of this literature study are applicable for any set of similar network discovery and network measurement tools (see Tables 2 and 3, respectively).

5. Conclusion

This literature study discussed a possible solution outline that engages with one of the major challenges for applying DSE in the context of dCPS: modeling the delay of complex network interactions between distributed subsystems. Related work about analytical network modeling, discrete-event simulation and network model validation was reviewed. A framework was created that approaches the network model creation step at multiple levels of abstraction to find the optimal speed-accuracy tradeoff balance, given a set of predetermined KPI goals. This framework also incorporates validation steps in every model iteration, for which network topology data and network measurements need to be collected.

Acknowledgments

This literature study was produced as part of the XM_0131 course at the Vrije Universiteit Amsterdam in cooperation with the University of Amsterdam. It was supervised by prof. Dr. Benny Akesson, professor by Special Appointment with the University of Amsterdam & Senior Researcher at TNO-ESI, and Faezeh Saadatmand, PhD candidate at the Leiden Institute of Advanced Computer Science.

References

[1] Ns-3 - a discrete-event network simulator for internet systems. https://www.nsnam.org/. Accessed: 2023-05-09.

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

[3] Ryan Berry. Queuing Theory. *Senior Project Archive*, pages 1–14, 2006.

[4] 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.

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

[6] 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.

[7] Svilen Ivanov, Andre Herms, and Georg Lukas. Experimental Validation of the ns-2 Wireless Model using Simulation, Emulation, and Real Network. January 2007.

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

[9] 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.

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

[11] 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.

[12] Levente Mészáros, Andras Varga, and Michael Kirsche. INET Framework. In Antonio Virdis and Michael Kirsche, editors, *Recent Advances in Network Simulation*, pages 55–106. Springer International Publishing, Cham, 2019.

[13] Jeonghoon Mo. *Performance Modeling of Communication Networks with Markov Chains*. Synthesis Lectures on Learning, Networks, and Algorithms. Springer International Publishing, Cham, 2010.

[14] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

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

[16] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, N.J, 1981.

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

[18] Robert G. Sargent. Simulation Model Verification and Validation. 1991.

[19] John F. Shortle, James M. Thompson, Donald Gross, and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley Series in Probability and Statistics. John Wiley & Sons, Hoboken, New Jersey, fifth edition edition, 2017.

[20] William J. Stewart. Performance Modelling and Markov Chains. In Marco Bernardo and Jane Hillston, editors, *Formal Methods for Performance Evaluation*, volume 4486, pages 1–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[21] 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.

[22] 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.

[23] 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.

Appendix

Name	Description	URL
AnyLogic	Simulation software used in various industries such as logistics, manufacturing, mining, healthcare, and more.	https://www.anylogic.com/
Cnet	Open-source network simulator for wired and wireless communication networks.	https://www.csse.uwa.edu.au/cnet/
EstiNet	Commercial network simulator with advanced features for modeling and simulating communication networks.	https://www.estinet.com/
J-Sim	Open-source network simulator with a focus on modeling and simulating wired and wireless communication networks.	https://www.kiv.zcu.cz/j-sim/
NCTUns	Open-source network simulator that supports various types of networks, including wired, wireless, and mobile.	http://nsl.cs.nctu.edu.tw/NSL/nctuns.html
NetSim	Commercial network simulator with a wide range of features for modeling and simulating communication networks.	https://www.boson.com/netsim-cisco-network-simulator
NS-2	Popular open-source network simulator for wired and wireless communication networks.	https://nsnam.sourceforge.net/wiki
NS-3	Open-source network simulator with a focus on modeling and simulating Internet systems.	https://www.nsnam.org/
OMNeT++	Extensible open-source network simulator with a modular architecture for modeling and simulating various types of networks.	https://omnetpp.org/
OPNET	Commercial network simulator with a wide range of features for modeling and simulating communication networks.	https://opnetprojects.com/opnet-network-simulator/
Riverbed Modeler	Commercial network simulator with a comprehensive set of features for modeling and simulating communication networks.	https://www.riverbed.com/products/riverbed-modeler/
QualNet	Commercial network simulator with a wide range of features for modeling and simulating communication networks.	https://www.keysight.com
SENSE	Open-source network simulator that focuses on modeling and simulating wireless sensor networks.	https://www.ita.cs.rpi.edu/
SimEvents	Discrete-event network simulator specifically designed for modeling and simulating event-driven systems.	https://www.mathworks.com/products/simevents.html
SimPy	Open-source discrete-event network simulator for modeling and simulating complex systems.	https://simpy.readthedocs.io/en/latest/
SIMUL8	Commercial simulation software for modeling and simulating various types of systems, including networks.	https://www.simul8.com/
SSFNet	Open-source network simulator with a focus on modeling and simulating wired and wireless communication networks.	http://helper.ipam.ucla.edu/publications/cntut/cntut_1501.pdf
SWANS	Open-source network simulator for wireless ad hoc networks with a focus on mobility and energy-awareness.	http://jist.ece.cornell.edu/

TABLE 1. A HIGH-LEVEL OVERVIEW OF THE STATE-OF-THE-ART IN DISCRETE-EVENT SIMULATORS

Name	Description	URL
Advanced IP Scanner	A free, fast, and easy-to-use network scanner that allows you to scan and analyze local and remote networks.	https://www.advanced-ip-scanner.com/
Angry IP Scanner	A popular open-source network scanner that scans IP addresses and ports to detect live hosts and obtain information about connected devices.	https://angryip.org/
Device42	A comprehensive network discovery and asset management software that provides automated network mapping, inventory management, and IT documentation.	https://www.device42.com/
Fing	A network scanning and monitoring tool that provides a comprehensive view of your network, including device discovery, network mapping, and troubleshooting capabilities.	https://www.fing.com/
Lansweeper	An all-in-one network discovery and asset management solution that scans and audits all devices on your network, providing detailed information about hardware, software, and licenses.	https://www.lansweeper.com/
LanTopoLog	A network discovery tool that provides network mapping and topology visualization features for managing and monitoring your network.	http://www.lantopolog.com/
Nagios	A powerful open-source network monitoring tool that provides comprehensive monitoring and alerting capabilities for networks, servers, and services.	https://www.nagios.org/
NetBrain	A network automation and documentation tool that offers network discovery, mapping, and troubleshooting capabilities, as well as automation for network tasks.	https://www.netbraintech.com/
NetCrunch	A comprehensive network monitoring and management solution that provides network discovery, mapping, monitoring, alerting, and reporting features for networks of all sizes.	https://www.adremsoft.com/netcrunch/
NetScanTools	A suite of network scanning and troubleshooting tools that provides a range of network discovery, mapping, and diagnostic capabilities for network administrators.	https://www.netscantools.com/
Nmap	A popular open-source network exploration and security auditing tool that provides host discovery, port scanning, and version detection capabilities.	https://nmap.org/
Open-AudIT	An open-source network auditing and inventory management tool that provides network discovery, mapping, and asset management features for IT professionals.	https://www.open-audit.org/
OpenNMS	An open-source enterprise-grade network management and monitoring platform that provides network discovery, mapping, monitoring, alerting, and reporting capabilities.	https://www.opennms.com/
PRTG Network Monitor	A powerful network monitoring and management tool that provides comprehensive network monitoring, alerting, and reporting features for networks of all sizes.	https://www.paessler.com/prtg
Solarwinds	A popular network management and monitoring suite that provides network discovery, mapping, monitoring, alerting, and reporting capabilities for IT professionals.	https://www.solarwinds.com/
SoftPerfect Network Scanner	A lightweight network scanning tool that provides host discovery, port scanning, and network mapping features for network administrators.	https://www.softperfect.com/products/networkscanner/
Spiceworks	An IT management and monitoring solution that provides network discovery, inventory management, help desk, and reporting features for IT professionals.	https://www.spiceworks.com/
Total Network Inventory	A network inventory and asset management tool that provides network scanning, software/hardware inventory, and reporting capabilities for IT administrators.	https://www.total-network-inventory.com/

TABLE 2. A HIGH-LEVEL OVERVIEW OF THE STATE-OF-THE-ART IN NETWORK DISCOVERY TOOLS

Name	Description	URL
AppDynamics	A commercial application performance monitoring (APM) tool that provides visibility into distributed systems.	https://www.appdynamics.com/
AWS X-Ray	A distributed tracing service that can be used to analyze and debug production environments.	https://aws.amazon.com/xray/
Datadog	A cloud-based monitoring platform that provides observability across infrastructure, applications, and logs.	https://www.datadoghq.com/
Dynatrace	A commercial APM tool that provides distributed tracing, metrics, and logs, as well as AI-powered root cause analysis and automatic problem detection.	https://www.dynatrace.com/
Google Cloud Trace	A distributed tracing service that can be used to monitor and troubleshoot distributed systems.	https://cloud.google.com/trace
Graphite	An open-source tool for collecting, storing, and visualizing time-series data.	https://graphiteapp.org/
Honeycomb	A distributed tracing and observability platform that provides real-time visibility into complex systems.	https://www.honeycomb.io/
Instana	A commercial APM tool that provides real-time monitoring and tracing of microservices-based architectures.	https://www.instana.com/
Jaeger	A popular open-source distributed tracing system that visualizes the timing and duration of requests in complex distributed systems to identify performance bottlenecks and optimize high-traffic environments.	https://www.jaegertracing.io/
LightStep	A distributed tracing system that provides real-time visibility into microservices-based architectures.	https://lightstep.com/
Netdata	An open-source monitoring tool that provides real-time metrics and visualizations of system performance.	https://www.netdata.cloud/
New Relic	A commercial APM tool that provides distributed tracing, metrics, and logs.	https://newrelic.com/
OpenTelemetry	A set of open-source tools and APIs for collecting, processing, and exporting telemetry data such as traces, metrics, and logs.	https://opentelemetry.io/
Zipkin	An open-source distributed tracing system that can be used to monitor and troubleshoot microservices-based architectures.	https://zipkin.io/

TABLE 3. A HIGH-LEVEL OVERVIEW OF THE STATE-OF-THE-ART IN DISTRIBUTED NETWORK MEASUREMENT TOOLS