CHAPTER 1

INTRODUCTION TO MODELING, SIMULATIONS AND ANALYSIS

Abstract

In this chapter, we have tried to provide a detailed overview of how MATLAB can be used for network simulation and modeling. We describe simulation and its various types, followed by their working principles and different terminologies, and the algorithms governing these simulations. Also described are various simulation software selections for MATLAB, the simulation tools based on high performance, followed by the different network models. This chapter will effectively help readers understand the concepts more clearly and provide them with a clear understanding of how to perform these tasks in MATLAB.

Keywords: MATLAB, modeling, simulations

1.1 MATLAB Modeling and Simulation

The representation of the system in learning is an abstraction, as mentioned previously. This means that it will not generally be considered with a more significant number of features as well as characteristics of the system and also that it only produces elected features and characteristics. Hence, based on simplifications and assumptions, the received model representation is a compact enactment of the system. Learning the sequence for identifying the appropriate entities and their relationships with the system is known as modeling.

Two important questions arise for any researcher when dealing with the evaluation of the performance of the system:

- *Question 1* - What is a good model?
- *Question 2* - How to obtain it?

Here it is essential to take care of two additional factors if the evaluation method is represented as the computer simulation.

(i) It has to be implemented in software once the performance model is built. Hence, the "model" performance is to be of high quality, as the implementation needs to represent it.

(ii) Hence, a suitable tool requirement needs to be selected which formally suits the process of the evaluation method. The modeling for computer simulations of these four cornerstones are combined. Sadly, the problem is that a large number of them are complex, and generally, we need specific prior learning experience with modeling and simulation.

"Essentially, all models aren't right, yet some are helpful" is the reality of execution. By remembering this, a great execution (*either for investigation or for reenactment*) has the following qualities:

- *Simplicity*: It is conceivable that the execution models are great. This does not imply that an execution model ought not be point by point or ought not to attempt to consider complex connections. In any case, a great execution displays its multifaceted nature when it fills the need of the assessment (*additionally alluded to as objective of the investigation*).

Because of the shortage of exactness in contrast with the real world, this is a fundamental point in reproduction models which are regularly scrutinized. This is basically the notion od computer simulation and manufactured the rganized nodes so that mixing of nodes can be minimized to get better accuracy. For the reusability of execution models, it has certain additional results. Usually, the assessment contemplates have different objectives (primarily if they will be distributed to established researchers), so there is likely a contrast in the utilized execution models.

For systems, it clarifies a large number of open-source reproduction models, and for across-the-board organizing frameworks, it explains the nonattendance of prevailing or standard reenactment models to a certain degree. To fill all needs, there

is no reenactment shown. Any recreation demonstrate which is accessible has been planned given a specific assessment target. One should first check the first assessment reason which is served by the model, and it needs to reuse the particular reproduction display.

- *Reliability*: The basic part of execution models is their stable quality.

The clarification behind the evaluation of modeling has been made clear initially for better efficiency. In the resulting phase of the modeling system, the assumptions and reenactments ought to have been noticed, including its evolution phases. From the learning phases of the model to final output phase, model representation and its simulation plays an important role in finding the reliability of the resultant model. In the simulation part of the modeling, the recreation of the model is very much important for better analysis of the model. The analysis result states that there difference between evaluation procedure and final output and its application model. So reliability factor has to deal with all the above-mentioned analysis. So a genuine programming model can only result in better reliability of the system, as it is reliable for better execution, evaluation and final applications.

Notwithstanding the qualities of a decent execution display, a great reproduction model ought to have the accompanying attributes:

- *Proficiency*: As for the execution show up, it ought to be finished reasonably after a particular instant of time. In like way, running events of simulation is quick and different analysis results can then be visible.

- *Checked*: The executed reenactment model ought to be confirmed, i.e., the similarity between the execution shown and reproduction demonstrate that it more likely than not has been checked by different strategies.

Note that this progression is not quite the same as approving an execution or reproduction display.

- *Code Quality*: Reliability on reusability of the code can result in using a specific programming method which is responsible for avoiding confusion in system programming and further simulation for achieving the desired results.

- *Accessibility*: Reproduction model ought to be available with the end goal that different gatherings confirm and approve the model themselves. As expressed previously, an execution show isn't required to be as point by point as could reasonably be expected. Truth be told, finding the correct level of exactness for an execution show is very troublesome. A typical oversight in demonstrating is to put excessive detail into a model because of the need for both involvement in execution assessment or foundation information about the framework under investigation. Lastly, a great execution does not need to be all inclusive nor for the most part reusable.

1.2 Computer Networks Performance Modeling and Simulation

1.2.1 Computer-Based Models

Computer-based models are usually classified as follows [1, 2]:

- Deterministic vs Stochastic: A deterministic model predicts a specific output from a given set of inputs with neither randomness nor probabilistic components. A given input will always produce the same output given the same initial conditions. In contrast, a stochastic model has some inputs with randomness, hence the model predicts a set of possible outputs weighted by their likelihoods or probabilities.

- Steady-state vs Dynamic: A steady-state model tries to establish the outputs according to the given set of inputs when the system has reached steady-state equilibrium. In contrast, a dynamic model provides the system reactions facing variable inputs. Steady-state approaches are often used to provide a simplified preliminary model.

- Discrete vs. Continuous values: A discrete model is represented by a finite codomain, hence the state variables take their values from a countable set of values. In contrast, a continuous model corresponds to an infinite codomain. Therefore, the state variables can take any value within the range of two values. However, there are some systems that need to be modeled showing aspects of both approaches which bring about combined discrete-continuous models.

- Discrete vs Continuous time: In a discrete model the state changes can only occur at a specific instant in time. These instants correspond to significant events that impact the output or internal state of the system. In contrast, in a continuous model the state variables change in a continuous way and not abruptly from one state to another. Therefore, continuous models encompass an infinite number of states. Discrete models are the most commonly used for network modeling and simulation.

Simulations can be carried out following two approaches: local and distributed. Distributed simulation is such that multiple systems are interconnected to work together, interacting with each other, to conduct the simulation. In contrast, a local simulation is carried out on a single computer. Historically, the latter approach has been the most widely used to simulate computer networks, but the increasing complexity of simulations is fostering the importance of the former approach.

Figure 1.1 summarizes the modeling and simulation process. Behavioral information extracted from a real system is used jointly with system specifications and requirements. Based on these inputs, a system model is built and subsequently validated through simulation. Usually, several performance metrics are determined during simulations, which can be compared with results extracted from experimentations with the real system. If both are similar, the model is considered as valid, while if they are not, the system model must be corrected. Similarly, performance metrics

are used to determine if the system model fulfills the requirements, so the designed system can be refined and extended in a controlled way.

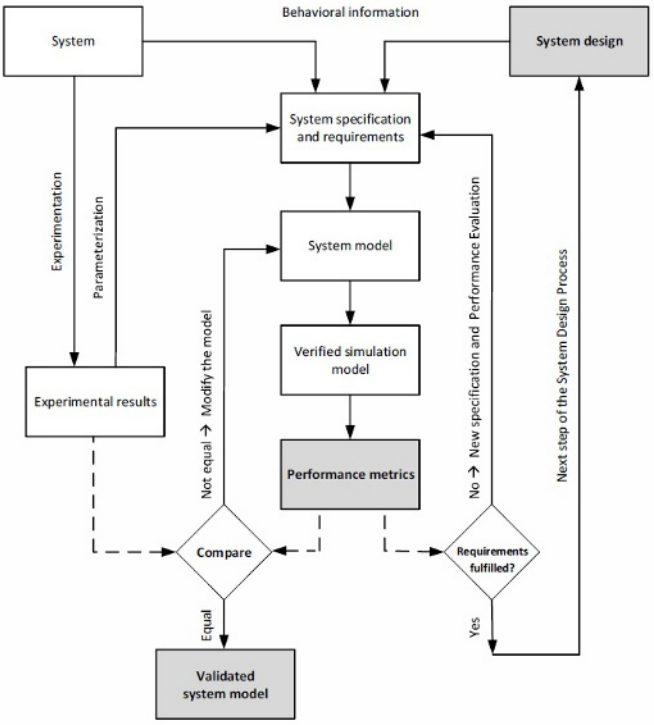

Figure 1.1 System modeling and simulation scheme.

1.2.2 Computer Network Simulation

Nowadays, computer network simulation is a fundamental element in network researching, development and teaching. Network simulators are widely used not only by practitioners and researchers, but also by students to improve their understanding of computer networks. Network simulation can be extremely useful when applied to scenarios such as protocol analysis, complex network deployment, evaluation of new services, prototypes or architectures, and so on. Broadly speaking, a network simulator can be understood as a black box, as illustrated in Figure 1.2 [13, 23, 24]

A network simulator implements a computer network model through algorithms, procedures and structures according to a given programming language. The implemented network model receives a set of parameters as inputs, which impose the rules and constraints of the simulation. In other words, these parameters set the course of the simulation. When the simulation process ends, the network simulator returns a set of results as outputs that can be used with a dual purpose: validating and veri-

fying the correctness of the implemented model for a particular domain of system states, and analyzing the behavior of the network modeled when the former has been successful [15-17].

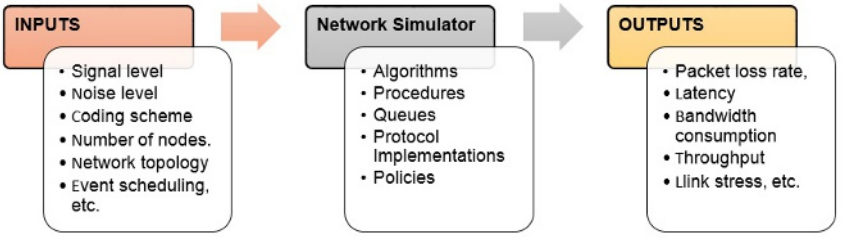

Figure 1.2 Network simulator abstraction.

Computer networks are most often simulated with discrete-event simulation. The main reason behind this widespread use is that discrete-event simulation adapts better to represent the behavior of computer networks, since computer network protocols can be modeled as finite state machines. In a computer network there is a steady state between two consecutive events, and discrete-event simulation allows for jumping from one steady state to another, leading to faster simulations. Other interesting aspects of discrete-event simulation are flexibility and lower computational overhead.

1.3 Discrete-Event Simulation for MATLAB

Discrete event simulation utilizes a mathematical/logical model of a physical system that portrays state changes at precise points in simulated time. Both the nature of the state change and the time at which the change occurs mandate precise description. Customers waiting for service, the management of parts inventory or military combat are typical domains of discrete event simulation [10, 11].

A more theoretical top-down description of discrete-event simulations is the following: The start of a discrete-event simulation is a model. A model is the construction of a conceptual framework, used to represent a system. The specific characteristics of discrete-event simulation model are the following:

1. Stochastic: At least some of the model's components are stochastic. This means that there is a certain factor of randomness in the model. This is represented by random number generators, which are explained in the terminology.

2. Dynamic: The time evolution of the system's components is important. Time is a significant variable.

3. Discrete: The state of the system changes only significantly when events occur at discrete time instances.

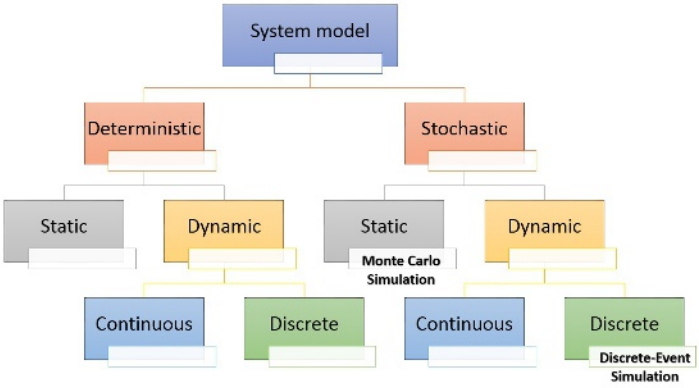

Figure 1.3 Discrete-event simulation.

The difference with continuous simulations is the use of the factor time. The system evolves over time so the variables change continuously in a continuous simulation. Whereas in a discrete event simulation the events themselves determine if something happens or not. Time can go by without anything happening.

We will give a concise introduction into discrete-occasion reenactment and for this we present the essential phrasing utilized in writing to clarify their connections. Likewise, for a discrete-occasion test system we present the center calculation, i.e., the time-propel occasion booking calculation.

1.3.1 Terminology and Components of Discrete-Event Simulation

Some common phrasing and elements for all discrete-occasion reproduction frameworks are known. In writing the terminology may change and surprisingly there is no institutionalized arrangement of terms. In a portion of our reference papers we freely adjust the definitions. A reflection of a specific subject of intrigue is alluded to as substance and it very well may be depicted by its own properties; For example, an element bundle carries its source (*address*), qualities, length as well as goal (*address*).

The relationship of a suitable set of entities is defined as *a system*. A certain purpose can be fulfilled by the relationship of a suitable set of entities, i.e., the system attempts to achieve a certain goal. For example, the goal of a network provides end-to-end connectivity while it may be defined by the routers, links and the entities hosts. The model is defined as the system and the use of the following methods refers to an abstracted system and is termed the model for clearness. During the process of minimizing the difficulty and the correlated cost as well as, effort involved, it will always consider the model in computer simulations.

1.3.2 The Principle of Discrete-Event Simulation

It is possible to utilize the discrete-event test system to bounce from one event to the next, and the rate an occasion might incite can be modified in the state of framework and also the age of a newer, supposed event occurred in future. To deal with every one of the events in the discrete-event simulation, the events are recorded as events seen later on the event list (FEL). To embed, to discover and to evacuate events seen, which are set later on, an event rundown ought to be actualized effectively later on in the event list [25].

Figure 1.4 demonstrates the growth of discrete-event simulation after some time. The events happen at the given time interval t_i and might alter the state of framework. To advance the reenactment with each discrete-event time t_i, in the PC memory a depiction of the framework is made that contains every required datum.

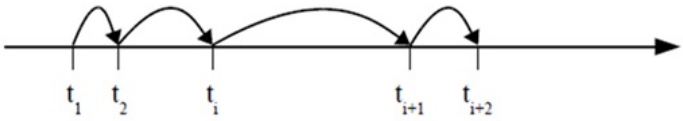

Figure 1.4 Principles of discrete-event simulation and the system state changes only at discrete point's t_i in time during the simulation.

Typically all discrete-event test systems share the following components:

- *System express*: The condition of the framework can be depicted by the arrangement of factors.

- *Entities*: Temporary entities are the elements that flow through the system. They wait in queues, follow a sequence of events and are specified by certain attributes. Most entities are introduced in a system according to a stochastic distribution. Examples of temporary entities are: parts of goods, customers and information messages. Permanent entities remain in the system when the simulation is executed. They are used in the events to process the temporary entities. The usage time of permanent entities is mostly stochastically distributed. Examples of permanent entities are: operators, servers and machines.

- *Attributes*: Entities own attributes that specify certain properties and states of the entity. They can affect how entities are handled during the sequence of the processes. They are not fixed through the flow so they can change by being processed. Examples of attributes are: priority and quality.

- *Clock*: The clock maintains the simulation time. The measurement units vary depending on the characteristics of the model. Because we are dealing with discrete-events, the time is not continuous but "jump" according to the events. The clock is also responsible for the timer of the events.

- *Queue*: Place where temporary entities wait an undetermined time before being accepted in an event. Some entities might have priority attributes to skip the queue. These queues can, for example, be used to locate overproduction or a bottleneck, as mentioned in the introduction.

- *Random number generators*: Most of the previous components need the possibility to generate random variables in order to simulate the stochastic distributions. Pseudorandom number generators are used to accomplish this.

- *Future occasion list*: To deal with the events, it very well may be appropriated by the information structure.

- *Statistical counters*: The result of a simulation should be shown in the form of a report that gives statistics about all the important components of the discrete-event simulation. These statistics are analyzed to make better decisions.

- *Initialization schedule*: To instate the reproduction display we utilize introduction routine and fix the clock value to zero in the framework.

- *Timing schedule*: From the future event list a routine recovers the the next occasion and also advances the clock value to the frequency in the framework.

- *Event schedule*: Events change the state of the system. A sequence of events can be bundled as an activity. These events determine the timing and the flow of the simulation, hence the word discrete-event simulation. A typical example of an event is the arrival of new entity, often used to start the simulation. The sequence of events ordered by time of occurrence forms an event list. Each simulation has at least one event list. This events list determines the mechanism for advancing the simulation time. This sequence of actions is also called event-scheduling. Every simulation also needs a stopping event or ending condition. This determines how long the simulation will run. Typical stopping events are a specific time in the future or a condition that the simulation itself fulfills. When a specific event happens amid the recreation is called an event schedule. By and large, an event routine is characterized for every event composed. In writing the term handler is frequently utilized synonymously.

1.3.3 ESTA Algorithm

An ESTA algorithm is an event-scheduling/time-advance algorithm [19]. According to their occurrence time, the event notifications are listed inside the future event list, i.e.,

$$fel = [t_1, t_2, ..., t_k] \text{ where } t_1 \leq t_2 \leq ... \leq t_k. \tag{1.1}$$

Here t_1 in this case is where an event occurs in time.

The diagram in Figure 1.5 below depicts the flow of the ESTA algorithm, which is subdivided into three main sections.

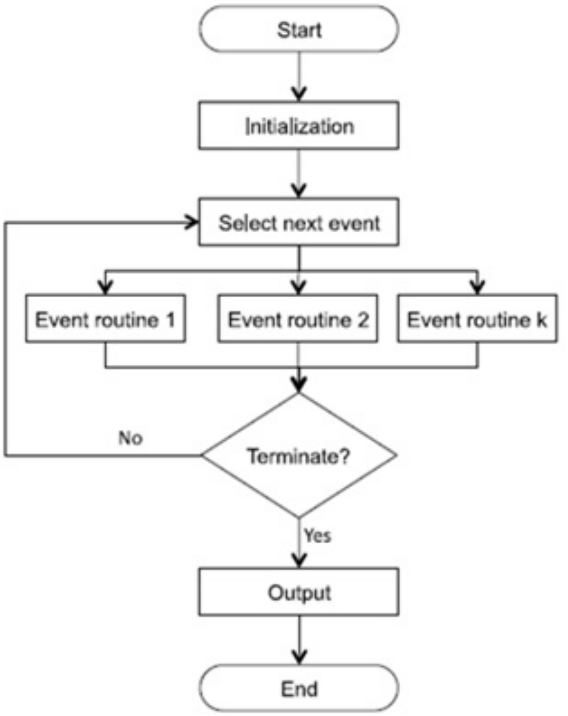

Figure 1.5 Flow chart explaining a discrete-event test system.

They are:

1. Section 1. Initialization
2. Section 2. Event Processing
3. Section 3. Output

Initialization employs three main processes: the first one is the clock, secondly, entities and finally, state variables. After that the simulator enters the second part and the events are processed in a loop. Here a type of specific event has been created, which is termed handler, in which current event is retrieved or recovered from the next events.

The state variables, entities and update statistics might be changed by the event routine and new event notices are created. The simulation enters the final part of the process after receiving the valid loop of termination condition. Finally, the statistics are calculated in the output part of the process and will be written into files if necessary.

1.3.4 ANALYSIS: Determination of Time to Attain Steady State Condition for MATLAB

By the end of fixed steady time final state will be achieved. You will achieve half of consistent state after one half a life, in the result we will achieve 75 percentage as unfaltering state after two half lives, and in the result we will attain 87.5 percentage enduring state after three half-lives. The enduring state will be attained after five half-lives, which is characterized in the thumb control (97% of consistent state accomplished). By utilizing a stacking dosage, without much effort you can attain the objective of an enduring state in the event that you have a medication with a long half life. For instance, it requires a 30 mg dosage once in multiple days on the off chance that you need to accomplish an unfaltering condition of fixation (Css) of 10 ng/mL.

By-and-by, to accomplish an unfaltering state it will take 10 days. Accept that the dosage proportionality, a 60 mg measurement, will accomplish a Css of 20 ng/mL in 10 days, you will attain half an enduring state after a solitary dosage of 60 mg (i.e., 20 ng/mL $\times$ half = 10 ng/mL). Subsequently, you can accomplish the relentless state level inside two days and, as previously stated, it could accomplish the objective Css quickly. The stacking measurement is 60 mg and the upkeep dosage is 30 mg. A critical term in pharmacokinetics is "consistent state," which can appear somewhat theoretical and confounding as well.

Another way to think of it is to envision a carton of eggs in your kitchen and you take two eggs out to make an omelet for breakfast and somebody sees that there is a vacant spot in the egg carton and purchases two more eggs to put in the carton. Hence, when you wake up the following morning, there is no empty spot in the carton of eggs and if this procedure repeats and again there is an adjustment made to the number of eggs in the carton, despite the fact that you utilize them for a few days, it will dependably be the same. For this situation you could without much of a stretch comprehend that the rate of end is equivalent to the rate of info and here the eggs are at relentless state.

1.4 Simulation Software Selection for MATLAB

It should be noted that the goal regarding various undertakings in mechanized manufacturing systems is to upgrade productivity and lower cost. To energize plans and review working techniques, the champion among the most typical methodologies is simulation because of the resultant variability and dynamic advantages of such model. Some of the various examinations to be conducted follow. There are certain suggestions made by Banks and Gibson when purchasing the diversion programming such as exactness and detail, incredible capacities, fastest speed, demo game plan of issue and comparing associations with practically identical applications by looking at accumulated social events of customers.

Together with their levels of essentialness for the specific reason, a purpose behind the development of propagation packages in the amassing region was presented by

Hlupic and Paul. While evaluating discrete-event diversion programming, a criteria for the structure should be considered, which was created by Nikoukaran *et al* [3]. A two-phase evaluation and selection methodology for simulation programming decisions was proposed by Tewoldeberhan *et al.* [4], in which the feature of propagation packages discussed above was empowered by the connection of two-phase evaluation in organized manner. For apparent improvement of each propagation package different methodologies are used in two-phase propagation evaluation.

For discrete-event multiplication, Seila *et al.* [5] showed a framework to pick the reenactment programming. The proposed framework attempts to recognize the endeavored target first by assessing around 20 programming instruments, since a run-of-the-mill understanding of the objective will help plot charts with internal association resources and also dealer and expert associations. A criterion which can help pros veritably assure business process organization tools was made by Popovi *et al.* [6].

By including the decision of propagation programming, these examinations are prepared with a more theoretical approach to the issues. This section discusses the programming model, followed by how to use it in the most efficient way, as it gives a couple of optimizing methods like representation of database model and the way the database has to handle the queries. Lastly, essential broadcasting of the analysis right from the inside of Smart Sim Selector can be derived and represented for future enhancement.

1.5 Simulation Tools Based on High Performance

A mobile ad hoc network (MANET) consists of several specific types of networks. One type is a wireless sensor network (WSN), shaped by several sensing devices which employ wireless transmission as the communication medium. It shares the same technical problems which have been noted in the research done on WSNs and MANETs. Two particular factors arise in WSN: usually one of the purposes of sensors is to measure the physical variables which are driven by the protocol layer operations and applications.

Hence, the network which governs various types of topologies and traffic load are sensed by the dynamics of physical parameters. In WSN, the primary concern is energy, and frequently the nodes will run on non-rechargeable batteries. Hence, as expected, the lifetime of a node is a fundamental element and must be taken into account. In actuality, in MANETs, vitality is an essential issue that ought to be enhanced, despite the fact that it is for the most part accepted that a hub can energize or supplant its battery. In the interim, sending and working a testbed to examine the real conduct of conventions and system execution is an awesome undertaking [7, 8]. Therefore, recreation is fundamental to contemplate wireless sensor network, since it is the preliminary process in testing newer applications as well as conventions in this field. For this reason, in reality this field has brought about an ongoing surge in reproduction apparatuses accessible to WSNs.

This scenario contains two key angles which ought to be assessed earlier than directed analysis: Firstly, its accuracy and, secondly, the reasonableness of executing a specific device. On the other hand, an expanding worry regarding strategy as well as presumptions regarding simulations have been presented by Aktar *et al.* [9] as follows: Admired equipment and conventions can prompt outcomes with mixed results. A "decent" model in view of strong presumptions is compulsory to infer trusted outcomes. In any case, including the required level of detail includes solid computational prerequisites. The essential trade-off is: Exactness and need for detail versus execution and adaptability. A device that constructs a model is required, and the client faces the task of choosing the fitting one. Reproduction programming normally gives a structure to demonstrate and recreate the conduct of genuine frameworks.

1.5.1 Network Model

Figure 1.6 shows a model of a wireless sensor network that depicts the general structure of the network which occurs on a wide scale [18].

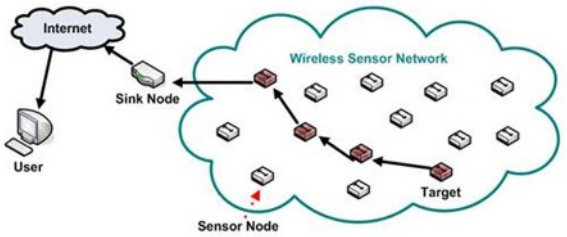

Figure 1.6 Wireless sensor network model.

The component descriptions are given below:

1. *Nodes*: Defined as a set of physical variables monitored by a physical device; each node refers to a physical device. Nodes can communicate with each other through a common radio channel. To control communications internally, a protocol stack is used. A sensor model is not similar to classical network models and it includes secondary components which involve the concept of physical node tier. In additional to it, a "topology" component may control node coordinates, which is not shown in Figure 1.6.

2. *Environment*: Additionally, an "environment" component is the major dissimilarity from the classical and wireless sensor network models. The nodes which are sensed by this component model trigger sensor actions along with the creation and growth of events, i.e., in the network, communication occurs between nodes.

3. *Radio channel model*: In the network, nodes signal propagation and can be characterized by the radio channel model. In ratio model, terrain components are used to calculate the growth of the radio channel.

4. *Sink nodes of wireless sensor network model*: Sink nodes are known as the special nodes in the network which receive the data present and process it. A note of interest is that they might be integrated in sensors to interrogate the sensors. Testing is performed by the simulator and the sink node used depends on the application.

5. *Agents*: Agents act as an event generator among the nodes which are present in the network. In the propagation through the environment and the sensor stimulation, the agent is also used to modify the physical magnitude.

Because of cross-layer interdependencies, node conduct relies upon collaborating factors. As shown in Figure 1.7, nodes are advantageous for partitioning a hub into theoretical levels.

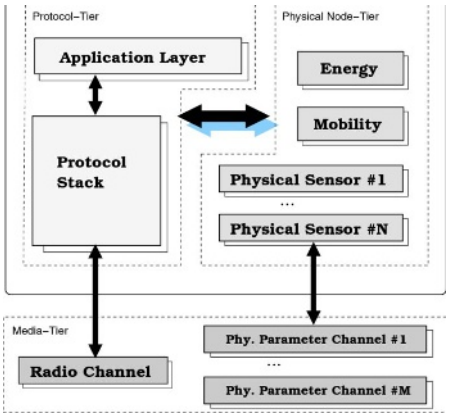

Figure 1.7 Tier-based node model.

- All the correspondence conventions are included in the protocol level. There are three layers which coincide with this level; for example, a directing layer, a MAC layer, and a particular application layer. In this way, in order to trade level data, a productive strategy must be created.

- The physical-hub level speaks to the equipment stage and its impacts on the execution of the instrument. An arrangement of physical sensors – the vitality module and the portability module – are known as normal components of this level. Physical sensors take the lead in checking equipment. An urgent issue of WSNs takes place at the equipment stage, where the vitality module mimics utilization management. Sensor position is controlled by the versatility module. A hub is associated through two conditions:

– Condition 1 - Associated through a radio channel
– Condition 2 - Associated through at least a physical channel.

1.5.2 Network Simulators

Some of tools of wireless sensor network are network simulators, while others are emulators. A network simulator is a tool used to evaluate network protocol performance in a wireless sensor network, it simulates the network. Some simulators that are discussed here are NS-2, Sensor Sim, SENS, and Em Star [12, 14, 20-22].

Table 1.1 Summary of the most important network simulators.

Nº	Simulator	Developer	URL	Commercial	Language
1	Cnet	Univ. Western of Australia	http://www.csse.uwa.edu.au/cnet	No	C
2	EstiNet	EstiNet Technologies	http://www.estinet.com	Yes	C++
3	GloMoSim	UCLA	http://pcl.cs.ucla.edu/projects/glomosim	No	C, C++
4	GTNetS	Georgia Tech Univ.	http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS	No	C++
5	IKR SimLib	Univ. of Stuttgart	http://www.ikr.uni-stuttgart.de/IKRSimLib	No	C++, Java
6	J-Sim	UIUC	http://j-sim.cs.uiuc.edu	No	Java, Perl, Tcl, Python
7	NCTUns	Nat. Chiao Tung *Univ.*	http://nsl10.csie.nctu.edu.tw	No	C++
8	NetSim	Tetcos	http://www.tetcos.com/netsim_gen.html	Yes	C++
9	NS-2	USC ISI	http://www.isi.edu/nsnam/ns	No	C++, OTcl
10	NS-3	ns-3 project	http://www.nsnam.org	No	C++, Python
11	OMNeT++	Technical Univ. of Budapest	http://www.omnetpp.org	No	C++, C#, Java
12	OMNEST	Simulcraft	http://www.omnest.com	Yes	C++
13	OPNET	Riverbed Technology	http://www.opnet.com	Yes	C, C++, Proto-C
14	QualNet	SCALABLE Network Technologies	http://web.scalable-networks.com/content/qualnet	Yes	C
15	SENSE	Rensselaer Polytechnic Institute	http://www.ita.cs.rpi.edu	No	C++
16	SimPy	MIT	http://simpy.readthedocs.org	No	Python
17	SSFNet	Renesys	http://www.ssfnet.org	No	C++, Java
18	SWANS	Cornell Univ.	http://jist.ece.cornell.edu	No	Java
19	SWANS++	AquaLab	http://www.aqualab.cs.northwestern.edu/legacy/swans++	No	Java
20	YANS	INRIA	http://sourceforge.net/projects/yans-netsim	No	C, C++

A network emulator is a tool used to evaluate real hardware performance on the nodes of a wireless sensor network. Some emulators that are discussed here are TOSSIM, ATEMU and AVRORA. Each of these classes and tools has its specific advantages and disadvantages and often the selection of the tool is mainly based on the experience of the researcher rather than on rational arguments. An overview of the different tools and simulation environments with their particular pros and cons has been established by the CRUISE project.The most relevant simulation environments used to study WSN are introduced and their main features and implementation issues are also described.

1.5.2.1 Network Simulator-2

Network Simulator-2 (NS-2)[1] is a discrete event simulator that was designed especially to simulate transmission control protocol (TCP) on wireless and wired networks and it was developed in C++. NS-2 is one of the most popular non-specific network simulators, and supports a wide range of protocols in all layers. It uses OTCL as configuration and script interface. NS-2 is the paradigm of reusability. It provides the most complete support of communication protocol models, among non-commercial packages. Regarding WSN, NS-2 includes Ad-Hoc and WSN specific protocols such as directed diffusion or SMAC. NS-2 can comfortably model wired network topologies up to 1,000 nodes or above with some optimizations. The drawbacks of NS-2 is that the learning curve to operate the simulator is considerable, and knowledge of a scripting language, queuing theory, and modeling techniques is required; also, because the protocols are written in the object-oriented language C++, the simulator does not scale well for simulations of large wireless sensor node networks, as each node is represented as an object and performance is significantly affected. Another disadvantage of NS-2 is that it provides poor graphical support, via Nam. This application just reproduces a NS-2 trace. NS-2 has been an essential testing tool for network research and, so, one could expect that the new conventional protocols will be added to future releases.

1.5.2.2 AVRORA

AVRORA[2] is another WSN simulator that simulates a sensor system at a very finely granular level; it was developed by Ben Titzer *et al.* [12] at UCLA. AVRORA attempts to achieve the advantages of TOSSIM and ATEMU, and mitigate these two emulator's disadvantages. While TOSSIM and ATEMU are written in the programming language C, AVRORA is written in Java. AVRORA emulates the node operation by running code instruction-by-instruction. The unique feature of AVRORA is how it handles the synchronization of nodes. There are two modes of synchronization: one uses a regular interval that can be set by the emulation operator to synchronize nodes; the other mode allows nodes to proceed along in simulation, until it reaches a state where it has to synchronize with its neighbors. Both of these synchronization modes reduce the overhead introduced by the synchronization function, as compared to TOSSIM and ATEMU.

[1] https://www.isi.edu/nsnam/ns/
[2] compilers.cs.ucla.edu/avrora/

1.5.2.3 OMNET++

OMNET++ (*Object-oriented modular discrete event simulator*)[3] is a modular discrete event simulator implemented in C++. Getting started with it is quite simple, due to its clean design. OMNET++ also provides a powerful GUI library for animation and tracing and debugging support. Its major drawback is the lack of available protocols in its library, compared to other simulators. However, OMNET++ is becoming a popular tool and its lack of models is being cut down by recent contributions. For instance, a mobility framework has recently been released for OMNET++, and it can be used as a starting point for WSN modeling. Additionally, several new proposals for localization and MAC protocols for WSN have been developed with OMNET++ under the Consensus project, and the software is publicly available. Nevertheless, most of the available models have been developed by independent research groups and don't share a common interface, which makes it difficult to combine them.

1.5.2.4 J-Sim

J-Sim (JavaSim)[4] is a component-based simulation environment developed entirely in Java. It provides real-time process-based simulation. The main benefit of J-Sim is its considerable list of supported protocols, including a WSN simulation framework with a very detailed model of WSNs, and an implementation of localization, routing and data diffusion WSN algorithms. J-Sim models are easily reusable and interchangeable, offering the maximum flexibility. Additionally, it provides a GUI (*Graphical user interface*) library for animation, tracing and debugging support and a script interface named Jacl.

1.5.2.5 NCTUns2.0

NCTUns2.0[5] is a discrete event simulator whose engine is embedded in the kernel of a UNIX machine. The actual network layer packets are tunneled through virtual interfaces that simulate lower layers and physical devices. This notable feature allows simulations to be fed with real program data sources. A useful GUI is available in addition to a high number of protocols and network devices, including wireless LAN. Unfortunately, no specific designs for WSN are included. On one hand, the close relationship between the simulation engine of NCTUns2.0 and the Linux kernel machine seems a difficulty (adding WSN simulation modules to this architecture is not a straightforward task). But, on the other hand, real sensor data can be easily plugged into simulated devices, protocols and actual applications, just by installing these sensors in the machine. NCTUns2.0 also has worthy graphical edition capabilities.

1.5.2.6 JiST/SWANS

JiST (Java in Simulation Time)/SWANS (Scalable Wireless Ad hoc Network Simulator)[6] is a discrete event simulation framework that embeds the simulation engine

[3] www.omnetpp.org/
[4] https://www.physiome.org/jsim/
[5] http://nsl.csie.nctu.edu.tw/nctuns.html
[6] http://jist.ece.cornell.edu/

in the Java byte code. Models are implemented in Java and compiled. Then, byte codes are rewritten to introduce simulation semantics. Afterwards, they are executed on a standard Java virtual machine (JVM). This implementation allows the use of unmodified existing Java software in the simulation, as occurs with NCTUns2.0 and UNIX programs. The main drawback of JiST tool is the lack of enough protocol models. At the moment it only provides an Ad-hoc network simulator called SWANS, built atop JiST engine, and with a reduced protocol support. The only graphical aid is an event logger. JiST claims to scale to networks of 106 wireless nodes with two and one order of magnitude better performance (execution time) than NS-2 and GloMoSim[7] respectively. It also has been shown that it outperforms GloMoSim and NS-2 in event throughput and memory consumption, despite being built with Java. Parsec is a simulation language derived from C that adds semantics for creating simulation entities and message communication on a variety of parallel architectures. Taking advantage of parallelization, it has been shown to scale to 10,000 nodes.

1.5.2.7 TOSSIM

TOSSIM (*Tiny operating system sensor network*)[8] is a bit-level discrete event simulator and emulator of TinyOS, i.e., for each transmitted or received bit an event is generated instead of one per packet. This is possible because of the reduced data rate (around 40 kbps) of the wireless interface. TOSSIM simulates the execution of nesC code on a TinyOS/MICA, allowing emulation of actual hardware by mapping hardware interruptions to discrete events. A simulated radio model is also provided. Emulated hardware components are compiled together with real TinyOS components using the nesC compiler. Thus, an executable with real TinyOS applications over a simulated physical layer is obtained. Additionally, there are also several communication services that provide a way to feed data from external sources. The result is a high fidelity simulator and emulator of a network of TinyOS/MICA nodes. The goal of TOSSIM is to study the behavior of TinyOS and its applications rather than performance metrics of some new protocol. Hence, it has some limitations; for instance, it does not capture energy consumption. Another drawback of this framework is that every node must run the same code. Therefore, TOSSIM cannot be used to evaluate some types of heterogeneous applications. TOSSIM can handle simulations around a thousand of motes. It is limited by its bit-level granularity: Performance degrades as traffic increases. Channel sampling is also simulated at bit level and consequently the use of a carrier sense multiple access (CSMA) protocol causes more overhead than a time division multiple access (TDMA) one.

1.5.2.8 EmStar/EmSim/EmTOS

EmStar[9] is a software framework to develop WSN applications on special platforms called microservers: Ad hoc systems with better hardware than a conventional

[7] networksimulationtools.com/glomosim-simulator-projects/
[8] networksimulationtools.com/tossim/
[9] https://www.usenix.org/legacy/publications/library/proceedings/usenix04/tech/general/full_papers/girod/girod_html/eu.html

sensor. The EmStar environment contains a Linux microkernel extension, libraries, services and tools. The most important tools are:

- EmSim: A simulator of the microserver's environment. In EmSim, every simulated node runs an Em-Star stack, and is connected through a simulated radio channel model. It is not a discrete event but a time-driven simulator; that is, there is no virtual clock.

- EmCee: An interface to real low-power radios, instead of a simulated radio model, obtaining radio emulation. EmStar source code (this code can be in any language) is used in the simulations.

- EmTOS: An extension of EmStar that enables nesC/TinyOS applications to run in an EmStar framework. Thus, it opens the way to heterogeneous systems of sensor and microservers. Simulation of microserver and sensor networks is also supported. In addition, EmTOS provides three modes of emulation: "Pure Emulation," where all the motes are emulated by software, "Real Mode," where all the motes are real, and "Hybrid Mode," where some motes are real and others are emulated. EmTOS reaches up to 200 motes and it is claimed that for over 500 nodes it would be necessary to distribute the simulation on several processors.

1.5.2.9 ATEMU

ATEMU[10], which stands for "Atmel Emulator," is a fine-grained sensor network simulator by Polley *et al.* ATEMU is described as a bridge between an actual deployment of a Wireless Sensor Network system, and a Wireless Sensor Network simulator, a hybrid simulator, where the operation of individual sensor nodes is emulated in an instruction by instruction manner, and their interactions with each other via wireless transmissions are simulated in a realistic manner [12]. Although it was not specifically designed for MICA2 motes and TinyOS, much of the research and testing of ATEMU was performed with this platform. It is an emulator of the AVR processor (this processor is used in the MICA platform). Although the operation of the mote is emulated instruction by instruction, the radio model is simulated. ATEMU also provides a library of other hardware devices, e.g., timers or transceivers.

Therefore, a complete hardware platform is emulated, obtaining two advantages:

1. The capability of testing OS and applications other than TinyOS.

2. The capability of simulating heterogeneous networks with different sensors.

They are achieved at the cost of high processing requirements and poor scalability. A key weakness to ATEMU is that it is significantly, approximately 30 times, slower that TOSSIM.

[10]http://www.hynet.umd.edu/research/atemu/

1.5.2.10 SENS

SENS is a Wireless Sensor Network simulator that was developed by the Open Systems Laboratory at the University of Illinois at Urbana-Champaign (UIUC) in 2004. SENS was developed as a customizable sensor network simulator for WSN applications. It has four main components that function to simulate the sensor node and network, and the environments in which the network might operate. There is an application component that simulates the software of the sensor node; and a network component that simulates the transmission of packets among the nodes. It operates in one of three modes: successful packet transmission, a probability of packet loss, and packet collision. The third component is the physical component that simulates the power, sensor, and actuator parts of the sensor node and interface with the fourth component, the environment. The environment component is simulated by considering how different surfaces affect radio wave propagation. The environment simulation component is the key benefit of SENS; it is otherwise less customizable than other network simulators. This discrete event simulator is implemented in C++. SENS utilizes a simplified sensor model with three layers (application, network and physical) plus an additional combined environment and radio layer. NesC code can be used directly on it.

1.5.2.11 PROWLER/JPROWLER

PROWLER/JPROWLER is a discrete event simulator running under MATLAB intended to optimize network parameters. JPROWLER[11] is a version of PROWLER (*Probabilistic wireless network simulator*) developed in Java.

1.5.2.12 SNAP

SNAP (*Simulation analysis platform*)[12] is a totally different approach. SNAP is defined as an integrated hardware simulation and deployment platform. It is a microprocessor that can be used in two ways:

1. As the core of a deployed sensor.

2. As part of an array of processors that performs parallel simulation.

Again, "real" code for sensors can be simulated. By combining arrays of SNAPs (called network on a chip), it is claimed to be able to simulate networks on the order of 100,000 nodes.

1.5.2.13 SensorSim

SensorSim[13] is another network simulator, which is based on NS-2. This simulator was developed in the network and embedded systems laboratory at UCLA. Similar to the model power that is used in the minimize usage extend life (MUEL) simulator, SensorSim enhanced NS-2 by having an advanced power model, taking

[11] https://w3.isis.vanderbilt.edu/projects/nest/jprowler/
[12] snap.stanford.edu/
[13] https://www.swmath.org/software/15014

into account all of the components on a sensor node that would use energy. Further, a significant advancement of SensorSim was to include, first a sensor channel, and later, a mechanism that allowed the simulation of external events that would trigger a reaction in the simulation. Another significant contribution of SensorSim was the development of a middleware software tool called sensor ware. This middleware functioned between the simulator and the simulated sensor nodes, and allowed dynamic management of nodes during simulation, such as the loading of scripts or other applications to the nodes. SensorSim, having evolved from NS-2, had similar problems of scalability, and is not publicly available today.

1.5.2.14 SSFNET

SSFNET (*Scalable simulation framework network*) [14] is a collection of Java SSF-based components for modeling and simulation of Internet protocols and networks at and above the IP packet level of detail. Link layer and physical layer modeling can be provided in separate components. SSFNet models are self-configuring; that is, each SSFNet class instance can autonomously configure itself by querying a configuration database, which may be locally resident or available over the Web. The network configuration files are in the DML format. They are used to configure a complete model, and instantiate a simulation with the help of the scalable DML configuration database package that is distributed with the SSF simulators. The principal classes used to construct virtually any Internet model are organized into two derivative frameworks, SSF.OS (for modeling of the host and operating system components, esp. protocols) and SSF.Net (for modeling network connectivity, creating node and link configurations). The frameworks SSF.OS and SSF.Net hide all details of the discrete event simulator SSF API, allowing the same implementation of protocols as in a real operating system.

1.5.2.15 Ptolemy

The Ptolemy[15] project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. A software system called Ptolemy II is being constructed in Java. The work is conducted in the industrial cyber-physical systems center (iCyPhy) in the department of electrical engineering and computer sciences of the University of California at Berkeley. The project is directed by Prof. Edward Lee. The project is named after Claudius Ptolemaeus, the second century Greek astronomer, mathematician, and geographer.

[14] www.ssfnet.org
[15] https://ptolemy.berkeley.edu/

1.6 Conclusion

Propagation is a principal tool to consider remote sensor systems in context of authentic analysis. This particular analysis suggests methods to detect recurrences in wireless sensor network and is also used for model representation every time the tool has been used. According to the concept of the models, OMNET++[16] (Object-oriented modular discrete event simulator) and SSFNET[17] (*Scalable simulation framework network*) limitations of open tradition models diverge from various test frameworks (*uncommonly, NS-2*), which constructs progression time. Mechanical assemblies like NCTU NS2.0 (Network Simulation 2.0), Linux operating system or Java platform are independently used in a propagation. At this stage, potential results immensely increase. In practice, mechanical assemblies, for instance, TOSSIM[18] (*Tiny operating system sensor network*), EmTOS or ATEMU, can reproduce certifiable sensor code.

In any case, late test framework shows that suitable software (*JiST/SWAN*) results in a performance that is higher than network simulation and GloMoSim[19] (in its back-to-back shape). Unmistakably, parallel simulations should perform and scale better than back-to-back ones. Parallel test frameworks, such as GloMoSim (*whose goal is execution instead of flexibility*), can reenact up to the limit of ten thousand remote center points. The standard aim of DASSF (*Digital access signaling system Fi*) parallel mechanical assembly is flexibility, with supportive topology arrangements as sweeping as hundred hundreds of wired parts. Graphical support is given by all the groups. OMNET++, NCTU NS2.0, J-SIM (*Simulation*) and Ptolemy software provide momentous GUI (*Graphical user interface*) libraries for development as well as investigation.

This chapter gave a clear explanation on the importance of simulation and how the process of simulation goes from scratch to the next level by implementing basic programming. We focused on explaining the whole concept with basic programming like Java and in some places we utilized Python code for making the models' design and implementing simple theories of the networking. We started with a discussion on MATLAB modeling and simulations and continued with a discussion on the types of simulations needed. Next, discrete event simulation using MATLAB was explained in brief. Then, the simulation software used for the modeling and simulation was explained and finally the tools used for high performance computing using network simulation methodology were explained.

[16] www.omnetpp.org/
[17] www.ssfnet.org
[18] networksimulationtools.com/tossim/
[19] networksimulationtools.com/glomosim-simulator-projects/

REFERENCES

1. Banks, J., & Gibson, R. (1997). Simulation modeling: some programming required. *IIE Solutions*, 29, 26-31.
2. Hlupic, V., & Paul, R. J. (1999). Guidelines for selection of manufacturing simulation software. *IIE transactions*, 31(1), 21-29.
3. Nikoukaran, J., Hlupic, V., & Paul, R. J. (1999). A hierarchical framework for evaluating simulation software. *Simulation Practice and Theory*, 7(3), 219-231.
4. Tewoldeberhan, T. W., Verbraeck, A., Valentin, E., & Bardonnet, G. (2002, December). An evaluation and selection methodology for discrete-event simulation software. In Simulation Conference, 2002. Proceedings of the Winter (Vol. 1, pp. 67-75). IEEE.
5. Seila, A. F., Ceric, V., & Tadikamalla, P. R. (2003). Applied simulation modeling. Brooks/Cole.
6. Popovi, A., Hackney, R., Coelho, P. S., & Jakli, J. (2012). Towards business intelligence systems success: Effects of maturity and culture on analytical decision making. *Decision Support Systems*, 54(1), 729-739.
7. Ahloowalia, B. S., Maluszynski, M., & Nichterlein, K. (2004). Global impact of mutation-derived varieties. *Euphytica*, 135(2), 187-204.
8. Ahn, S.; Mukelar, A. Rice Blast Management Under Upland Conditions. Progress in Upland Rice Research; Int. Rice Res. Inst.: Manila, 1986; pp 363-374.
9. Aktar, M. W.; Sengupta, D.; Chowdhury, A. Impact of Pesticides Use in Agriculture: Their Benefits and Hazards. *Interdiscip. Toxicol*. 2009, 2, 1-12.
10. Paluszek, M., & Thomas, S. (2019). MATLAB Graphics. In MATLAB Machine Learning Recipes (pp. 45-71). Apress, Berkeley, CA.
11. Henriksson, D., Cervin, A., & Arzn, K. E. (2003, October). TrueTime: Real-time control system simulation with MATLAB/Simulink. In Proceedings of the Nordic MATLAB Conference. Copenhagen, Denmark.
12. Pathan, A. S. K., Monowar, M. M., & Khan, S. (2014). Simulation technologies in networking and communications: selecting the best tool for the test. CRC Press.
13. Law, A. M., Kelton, W. D., & Kelton, W. D. (1991). *Simulation modeling and analysis* (Vol. 2). New York: McGraw-Hill.
14. Chang, X. (1999). Network simulations with OPNET. In WSC'99. 1999 Winter Simulation Conference Proceedings. *Simulation-A Bridge to the Future* (Cat. No. 99CH37038) (Vol. 1, pp. 307-314). IEEE.
15. Jain, R. (1990). The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. John Wiley & Sons.
16. Law, A. M., Kelton, W. D., & Kelton, W. D. (2000). *Simulation modeling and analysis* (Vol. 3). New York: McGraw-Hill.
17. Jain, R. (1990). The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. John Wiley & Sons.
18. Shah, R. C., Roy, S., Jain, S., & Brunette, W. (2003). Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2-3), 215-233.

19. Fortier, P. J. (2018). Modeling and Analysis of Local Area Networks, CRC Press.
20. Sommer, C., Eckhoff, D., Brummer, A., Buse, D. S., Hagenauer, F., Joerer, S., & Segata, M. (2019). Veins: The open source vehicular network simulation framework. In Recent Advances in Network Simulation (pp. 215-252). Springer, Cham.
21. Aktas, I., King, T., & Mengi, C. (2010). Modeling application traffic. In Modeling and Tools for Network Simulation (pp. 397-426). Springer, Berlin, Heidelberg.
22. Frasconi, P., & Smyth, P. (2003). Modeling the Internet and the Web: probabilistic methods and algorithms. Wiley.
23. Burbank, J. L., Kasch, W., & Ward, J. (2011). An introduction to network modeling and simulation for the practicing engineer (Vol. 5). John Wiley & Sons.
24. Surez, F. J., Nuo, P., Granda, J. C., & Garca, D. F. (2015). Computer networks performance modeling and simulation. In Modeling and Simulation of Computer Networks and Systems (pp. 187-223). Morgan Kaufmann.
25. Guizani, M., Rayes, A., Khan, B., & Al-Fuqaha, A. (2010). Network modeling and simulation: a practical perspective. John Wiley & Sons.