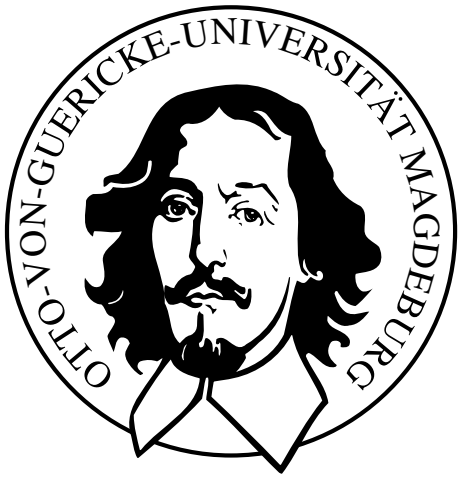
Distributed Simulation of Wireless Networks with ns-2

– Master's Thesis –

Svilen Ivanov



Otto-von-Guericke University – Magdeburg
Faculty of Computer Science
Institute for Distributed Systems

Supervision: Prof. Dr. Edgar Nett
Dipl.-Inf. Daniel Mahrenholz

Abstract

Ns-2 is a widely used simulation tool for computer networks. However, wireless network simulations are time-consuming because ns-2 is a sequential simulator and models wireless networks up to a high detail. The long simulation runs are inconvenient for the development of communication protocols, because they prolong the development cycle. This motivates the goal of the thesis to speed up wireless network simulations by using parallel and distributed simulation techniques. The task of this thesis is to integrate a distributed simulation system in ns-2, that allows to simulate wireless networks on multiple computers within one network.

This thesis develops an extension to the ns-2 to simulate wireless networks with stationary nodes in a parallel and distributed fashion. The solution uses graph partitioning techniques to automatically separate the model, which has a size from 100 to 1000 nodes. Then, the model is run on a cluster of interconnected computers using distributed synchronisation methods. The solution of the thesis allows also parallel simulations on multiprocessor computers with the same implementation. Experimental results show that the task of the thesis is solved. Parallel and distributed simulations are correct and produce the same results as sequential ones. However, the solution does not reduce the running time of the simulations in the general case, because of the high amount of time spent on management of the distributed simulation. Speedup is achieved only in special cases, which require a minimum management overhead.

This thesis shows that the resulting speedup highly depends on the used knowledge about the simulation model. Additional investigations are needed in order to use a higher level knowledge for the management of parallel and distributed simulation. Experimental results show that the use of this knowledge would speed up parallel simulations of wireless networks.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BMBF	German Ministery of Education and Research
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CTS	Clear To Send
DCF	Distributed Coordination Function
DECT	Digital Enhanced (former European) Cordless Telecommunications
DES	Discrete-event simulation
DFG	Deutsche Forschungsgemeinschaft (German Research Foundation)
GEA	Generic Event Application Programming Interface
IEEE 802.11	Standard for wireless local area networks
IFS	Inter-frame Space
ISM	Industry Science Medicine
LP	Logical Process
MAC	Medium Access Control
NAV	Network Allocation Vector
NS-2	Network Simulator 2
PCF	Point Coordination Function
PHY	Physical layer of the OSI Reference Model
RNG	Random Number Generator
RTS	Request To Send
RUDP	Reliable User Datagram Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
WIGWAM	Wireless Gigabit With Advanced Multimedia Support
WINDECT	Wireless Local Area Network with DECT Telephony
WLAN	Wireless Local Area Network
XTP	Express Transport Protocol

1 Introduction

1.1 Context of the Thesis

Simulation is a development technique that is widely applied in many areas of research and industry. It is used to predict the operation of a real or imaginary system by inspecting an artificial representation of this system, called model. The benefit from simulation is knowledge about the behaviour and properties of the system, that can be used for different purposes.

One application of simulation is to predict the evolution of an existing system. Here the question is how a real system is going to evolve in the future. Simulation uses a current status of the system in order to predict future state(s) before the system evolves and reaches them in reality. In this case simulation gives information about the future. A typical example here is the weather forecast.

Another application of simulation is the development of a completely new system. Then, a model of an imaginary system can be used to test its design before the system is constructed in reality. This method can safe time and resources in the development process. Consider that a government plans to build a railway transportation system on the territory of its country. This is a complicated, time consuming and expensive process and therefore requires a thorough planning before it is started in reality. A simulation can help here, because it allows to easily modify a model and test different designs of the system. Another advantage is that simulation can test the system under reproducible conditions. This facilitates the planning, because different designs can be tested under the same conditions in order to choose the best one. The repeatability of simulation also helps to locate and correct errors in the design or the implementation.

Another important application of simulation is by a modification of an existing system. If an existing system has to be modified, while it is in operation, a simulation may be useful to predict the effects of the changes on the system. Consider that the government wants to extend the railway transportation system by an additional track. Then, they have to first analyse how the new track and the traffic on it will influence the current system, and whether existing timing plans for trains can be kept, or they need a new schedule. Since the railway system is already in operation it is useful first to test the modified system in a simulation, and then realize the changes in reality. This can avoid design errors and reduce the outage times during the realization of the changes.

In the context of this thesis, simulation is applied in the second and the third case. It is used to design a completely new system, which can be later additionally modified. However, as the title of the thesis implies, the context are not railway systems, but computer networks. Computer networks, like the transportation systems are also complex, and their proper operation requires a comprehensive planning.

Wireless networks are a technology that allows users to communicate over-the-air

without installation of wires. They have shown a continuous growth in the last years. This is mainly because they are easier to install than the wired ones. Another big advantage is that they support the mobility of the users. At the beginning the wireless networks have been designed primary for pure data transfer without requirements for high bandwidth and interactive applications. But as their size and popularity increased the user requirements also increased. Currently the application requirements on wireless networks raise continuously and approach the requirements on the wired ones. These are for example multimedia applications like video conferencing, which require a high bandwidth. Another examples are time critical applications like telephony over wireless networks. They require a timely delivery of voice data in order to achieve a good speech quality. Another application is the coordination of a fire brigade during operation via a wireless network. Since this scenario does not provide an existing network infrastructure the firemen have to build a network on-the-fly (ad-hoc) in order to communicate and synchronise their actions. Also, it might not be possible for every two firemen to communicate directly through a wireless network. They might be spread over a large distance, or between the walls in a building. So, the communication devices have to organise a multihop ad-hoc network and provide a transparent connectivity to the firemen. The devices may act both as end stations, and as routers for other communication flows.

However, wireless networks have many different features than the wired ones. These are mainly higher error rate, lower bandwidth, signal interference with other networks, highly dynamical topology. These properties have in most cases a negative impact on the applications and they do not perform as well as in wired networks. Therefore the research in wireless networks nowadays increases significantly. One example is the WIGWAM project, funded by the German Ministry of Education and Research (BMBF). Its goal is to develop a complete system for wireless communications at an ultra-high data rate of 1GBit/s to support multimedia applications[16]. Another example is the EU research project WINDECT, which is addressed to the time-critical voice applications. Its goal is to integrate a professional quality telephony into wireless networks [10].

This thesis is developed in the group of Real-time Systems and Communications at the Computer Science faculty at the Otto-von-Guericke University in Magdeburg. This group also takes part in a research project on wireless networks. It is a DFG (German Research Foundation) priority program under the name: “Middleware for Self-organising Infrastructures in Networked Mobile Systems”. The goal of the program is to develop methods and techniques for communication middleware in mobile multihop ad-hoc networks. The project at the Otto-von-Guericke University is called: “A Publisher/Subscriber-based Middleware with Quality of Service Guarantee to Support Mobile Applications”. Its goal is to develop transport and routing protocols which guarantee fault tolerance and timely correct delivery in multihop wireless

networks. The size of the target networks is in the range from 100 to 1000 nodes[3]. A typical application scenario is the coordination in a fire brigade, described above. Another application is to locate services in a mobile wireless network, based on their names. For example, this provides to the applications a service to use the nearest printer, or the nearest Internet gateway without considering the multihop and dynamic structure of the network.

Like the development of a transportation system, the design and implementation of a communication protocol has a complicated nature. A protocol has a similar design to an algorithm, but an algorithm usually operates in one unit, while a protocol is always run by multiple units at the same time, which interact with each other. So, the design of a protocol requires to think *in parallel*, and on behalf of multiple units in one network. This parallel thinking is difficult for a single mind of a human. Therefore a design of a high-requirement protocol only with a flowchart like an algorithm can hardly predict all possible situations and program flows.

Therefore, the development and testing of the communication middleware at the group of Real-time Systems and Communications goes through a simulation phase. Simulation is used to obtain information about the protocol, and to test whether it satisfies the requirements before running it in real networks. Simulation helps because the target networks are relatively big and hard to build and maintain during the development. Moreover, simulation is a convenient development framework, because it allows easily adjustable and reproducible testing conditions. This facilitates the evaluation of the middleware in different environments and the test and debugging of its operation.

The development cycle of a communication protocol in our group goes through the following phases. First is the definition of requirements to the protocol, i.e. which services it should provide and with which quality. Then the protocol is designed and implemented in a simulation, with the goal to satisfy the requirements. Then the protocol is statistically evaluated to test whether it satisfies the requirements. If the protocol does not satisfy the requirements, it is again redesigned, this time with more information from the tests, and again implemented. This is a continuous cycle, which aims to improve the quality of the protocol with each iteration.

During the second stage of this cycle (the implementation phase) the simulation is periodically run in order to test whether the protocol provides delivery of packets. This simulation aims to show whether the implementation is operational and the protocol provides connectivity in the network. It has to answer the question *whether* the simulation is operating, and not *how* it is operating. Therefore statistical analysis is not needed at this step. The protocol is analysed statistically after its design is completely implemented, which completes a single iteration in the development cycle.

The simulation tool, that is used at the group of Real-time Systems and Communications is the network simulator ns-2[9]. It is a discrete-event network

simulator that can simulate a wide range of networks, including wireless ones. Ns-2 is freely available, widely used and continuously developed in research and industry. It makes a good approximation of the lower wireless network layers (Data Link, Physical) and is well-suited for development of network protocols at the higher layers (Network, Transport). Furthermore there are extensions of ns-2, developed in our group, which facilitate the development of communication protocols. They are part of the GEA (Generic Event API) middleware, which allows to use the same implementation of a protocol both in simulated and in real networks[20]. Therefore ns-2 is tightly coupled with the development in our group, and actively used for wireless network simulations.

1.2 Motivation for the Thesis

The simulations of wireless networks with ns-2 are time-consuming. This is due to the detailed simulation of the lower network layers, the broadcast nature of wireless networks and high number of nodes (100 ... 1000). Ns-2 models the data link and physical layers up to a high detail — according to the IEEE specifications for wireless networks[22]. The high number of nodes and the broadcast nature of wireless networks further increase the complexity of the simulation. For example, when a node sends a packet, all other nodes within a given range have to be notified about it, which increases the number of computations quadratically.

Another reason for long-running simulations is that ns-2 is a *sequential* simulator. This means that it uses a single processing unit to execute the model. Ns-2 in its standard version can not run in parallel on multiple processors.

Figure 1.1 represents a measurement of the running time of the simulator. It shows the needed running time for one second simulation, depending on the number of nodes. The model is a wireless ad-hoc network in ns-2, running a simple polling protocol. The hardware platform, used to run the simulation, is a laptop with a 1.4 GHz Pentium M processor.

When the number of nodes is in the range 100...200 one second simulation time takes nearly 1 second running time. When the number of nodes increases up to 500, the running time of the simulation is bigger than the simulation time by almost a factor of 10. This means that a single simulation run of 3 minutes would take almost half an hour to complete. Three minutes is a relatively short time for the operation of a network. But periods of this magnitude are used during the development phase to test the implementation of the model.

The long simulation runs are inconvenient during the implementation phase of the model. During this phase the simulation engineer performs a single simulation run after each change in the model. The goal of this run is to show whether the model is operational or not. The long running time is inconvenient because the developer has to wait long time after each minor change in the model. This waiting time increases the time for one development cycle and leads to less iterations during the

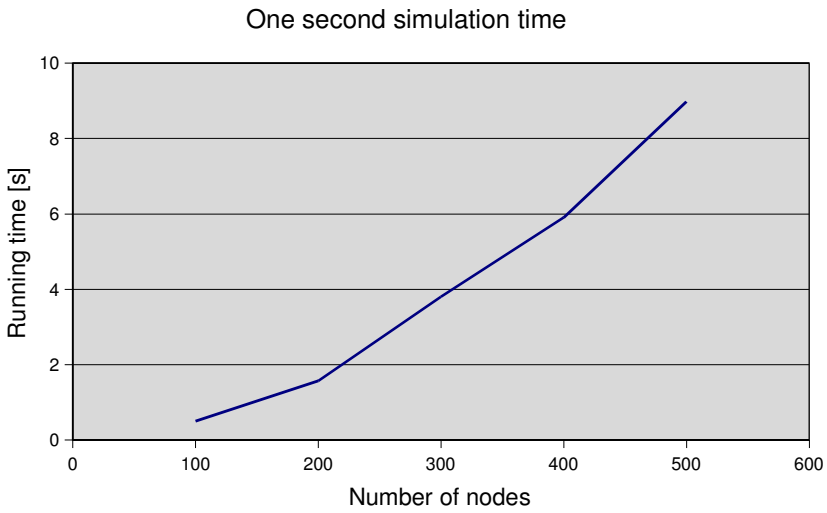


Figure 1.1: Simulation running time, depending on the number of nodes

whole development process. On the other hand each iteration through the development cycle possibly increases the quality of the communication protocol.

So, the wish to increase the quality of communication protocols have motivated a wish to decrease the running time of simulations.

1.3 Goal of the Thesis

The goal of this thesis is to reduce the running time of simulations with the network simulator ns-2. The reduction of running time should be achieved by using multiple processors, that work together to complete the *same* simulation task. These processors can be either single workstations, connected by a network, or parts of a multiprocessor machine.

The goal of the thesis can be more exactly defined, by using the term *speedup*. Speedup is a relation between the running time of one simulation in a sequential simulator T_{seq}, and the running time of the same simulation in a parallel/distributed simulation T_{par}:

$$Speedup = \frac{T_{seq}}{T_{par}}$$

The speedup is *positive*, when it is greater than one. This means that the parallel or distributed simulation runs faster than the sequential one. The speedup can be theoretically maximum the number of processors in a distributed simulation, but in practice it is lower, because the organisation and control of a parallel/distributed simulation also takes processing time. If this organisation and control takes too much

time it is possible that the speedup becomes *negative*. This means that the speedup is less than one. In this case it is not appropriate to use a parallel/distributed simulation, because the sequential one is faster.

The goal of this thesis is to achieve a positive speedup from distributed simulations of wireless ad-hoc networks with the ns-2. The goal can be defined explicitly by the expression:

$$Speedup > 1$$

1.4 Task Definition

The task of this thesis is to find and implement a distributed way of execution of the model in ns-2. The model should run on a cluster of computers, connected by a Fast Ethernet network.

The task is divided into the following major subtasks:

1. Automatic Partitioning

   Automatic partition of the simulated network is a preparatory step for the distributed execution. Due to the big size of the target networks (100...1000 nodes) the partition can hardly be done by hand. So, this first subtask has to divide the model of the initial network into sub-models, which can be executed in parallel. The resulting partition should maximise the speedup, gained by the distributed execution.

2. Distributed Simulation

   Extend the ns-2 to simulate a wireless ad-hoc network in a distributed fashion. This is the main part of the work. The task at this step is to execute the model on multiple processors and provide the same results as when the model is executed sequentially. The main challenge is to maintain the integrity of the model in the distributed execution. A requirement to this subtask is to provide a possibility for parallel execution of the model. This is a possibility to use a multi-processor machine to execute the model instead of a network cluster.

Since the complexity of the task is relatively high for a master's thesis, a simplification of the problem is made in order to reduce it. The simplification is that the network nodes in the model do not move during the simulation. The communication method is still ad-hoc, but the nodes keep their initial positions. The movement of nodes increases the complexity, because it introduces dynamics in the model and in the simulation infrastructure. If the nodes move, periodical re-partitions and re-organisation of the distributed simulation should be done, which increases the complexity of the design and implementation.

Note that the fastest speedup of a simulation run can be achieved by using independent replications of the model on different machines in parallel. This method runs the same model sequentially and independently on different machines, but with different random number streams. Independent replications are used for statistical analysis of the simulation results[14].

This work is aimed to speedup a single simulation run and should not be used for statistical analysis. It is designed for the implementation phase of the model due to the frequent changes and test runs in this period.

1.5 Document Structure

This thesis has the following structure. Section 2 defines terms and gives background information, used in the rest of the document. After that, section 3 discusses works, related to distributed simulation of wireless networks. Section 4 explains the solution of the task on an abstract level. It is concentrated on the design decisions taken to complete the thesis task. Section 5 describes technical details of the implementation. Then, section 6 describes experiments that test and evaluate the solution. Finally, section 7 concludes the thesis and gives possible directions for future research.

2 Fundamental Knowledge

This part of the document introduces basic terms and definitions used in the work. It discusses topics, which are a base for the rest of the thesis. Sections 2.1 and 2.2 are a short introduction of Discrete-Event Simulation (DES). They describe the principles of sequential and distributed DES. Section 2.3 describes the basic operation of wireless IEEE 802.11 networks. The discussion is focused on the properties, represented in the model of ns-2, and relevant for their simulation. Section 2.4 discusses the basics of the network simulator ns-2 and concentrates on the operation of the WLAN model. Finally, section 2.5 discusses properties of wireless networks, used for parallel and distributed simulation.

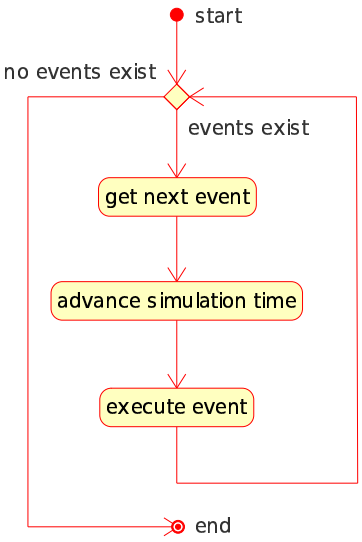
2.1 Sequential Discrete-event Simulation

Parts of this section are based on a lecture, and a book for computer simulation [14, 21].

Simulation *Simulation* is an imitation of the action of a real system and its development over time. Simulation generates an imaginary history of the system, which is used to inspect and evaluate the system. Simulation is based on a *model*. A model is a representation of the real system, which consists of *variables* and *assumptions* for its operation. The variables correspond to values of interest in the real system, e.g a number of collisions in a computer network. The assumptions represent dependencies, actions and development of the system. They are usually based on the variables and can be analytical (equations or relations), algorithmic (description of operations), logical (conditions). The model is run in order to analyse/predict the operation of the real system. The points of interest are usually the variables and statistics based on them. In network simulation these can be the number of packet transmission errors or the average packet transmission time. A simulation is called *discrete-event* when the changes in the model happen in distinct moments in time. These moments are defined and calculated by the model. No changes occur in the model between two such consecutive moments. Discrete-event simulation can be used to simulate computer networks. They have a discrete nature since they progress stepwise e.g. "join a network", "receive data", "send data".

Event The base term in discrete-event simulation is *event*. An event is an activity that happens in the model at a given moment in time. It represents an action that occurs in the real system being modelled. An event may change model variables and schedule another event(s) that happen as a consequence. One example is the "send data" event. It marks one data packet as sent and schedules a "receive data" event on the destination host after a period of time. This period is the packet transmission time, that is calculated by the model using its assumptions.

Algorithm 2.1 Algorithm of discrete-event simulator



Simulation Time One of the first abstractions, made in building DES models is the abstraction of time. *Simulation time* is a virtual time used to run the simulation model. Since no changes occur between two consecutive events the simulation time evolves also stepwise. It jumps over the moments at which the events happen. So, the algorithm of a discrete-event simulator can be described on figure 2.1.

The algorithm is relatively simple. The events are stored in a list ordered by their timestamp. The simulator takes the next earliest event from the list, advances its virtual simulation time to the event's timestamp and executes the event.

Random Variables Another abstraction usually made in the simulation models is the use of *random variables*. They are used to model a given random phenomenon in the real system by a sequence of pseudo random numbers with a similar distribution. A classical example here is the time, that a server needs to perform a task. It is usually modelled by a random variable which has an uniform distribution with parameters the mean value μ and the variance σ^2. In simulation, the time in which a server performs a task is determined by taking a pseudo random number from the used distribution. These pseudo random numbers are generated by random number generators (RNG). A RNG is an automaton that produces a sequence of numbers, that seem to be random. These sequence depends on a starting value, called *seed* of the RNG. If the same seed is used twice, then the RNG produces the same sequence of random numbers. So, the result of the simulation remains the same i.e. the simulation is *reproducible*. But when the seed is changed, the sequence also changes, which leads to a different simulation result. The method of *independent replications* is used to obtain

a *statistically significant* simulation result [14].

Event Relations An important relation between events in sequential DES is the *affect*. An event A *affects* event B if the following conditions hold:

1. A occurs before B, and
2. A modifies model variables that B uses, i.e. the changes that A makes to the model are relevant for B.

In this case it is also said that B *depends* on A. This relation is transitive, e.g. if A affects another event A' and A' affects B, then A also affects B. If A does not affect B and B does not affect A, then A and B are called *independent*. Based on the independence we can define the following lemma, which is important for distributed DES:

Lemma Two events can be executed in *any* order if they are independent and have different timestamps.

Proof Let A and B be two independent events and A has the smaller timestamp. Then A can be executed first, because it should happen before B. B can also be executed first, because A does not affect it (from the independence), i.e. it is irrelevant for B whether A has been executed or not. The proof is similar if B has a smaller timestamp.

This lemma is very useful in distributed DES, because it allows to execute independent events in parallel.

2.2 Distributed Discrete-event Simulation

Distributed Simulation *Distributed DES* is a discrete-event simulation, that is executed on many interconnected computers at the same time[18]. These computers are connected by a network, and may be spread over small areas (e.g. a room, or a building), or large areas (e.g. Internet). On the other hand, *Parallel DES* is a discrete-event simulation, that is run concurrently on many processors in a multiprocessor environment. A difference between parallel and distributed simulation is that distributed DES implies higher communication latency than parallel DES. This is because computer networks usually have much higher latencies than shared memories. This section discusses distributed simulation, but most of the terms are also valid for parallel simulation.

Logical Process A basic term in parallel/distributed simulation is *logical process* (LP) [18]. It is a representation of a physical process in the real system, being modelled. The real system is viewed as a composition of physical processes that interact in some way. In the model of this system each physical process is represented by a logical process. The interactions among the physical processes are modelled by message exchange among the logical processes. Lets take a computer network as an example of a real system. Then we can consider the network nodes as physical processes and the packets that these nodes exchange as the interactions. In the model each network node is represented by a logical process, and the network packets by messages that the LPs exchange. Note that the notion of a physical process in the real system can be defined by the modeller. We can consider also a group of nodes (a subnetwork) as one physical process and the packets that flow among subnets as the interactions. Then each LP represents a subnetwork and the packets between the subnetworks could be modelled by exchange of messages between the LPs.

In the context of this work a logical process (LP) also denotes the simulator executive, that models a physical process in the real system (network). In this sense the phrases "synchronisation among LPs" or "message exchange among LPs" mean interactions among the simulator executives, running a parallel/distributed simulation.

Decentralized event level distributed DES A discrete-event simulation can be distributed at different levels [15]. This means to choose properties of the simulation model, that will be used to execute it in a distributed way. Next, I focus on the decentralized event level distribution. Section 4.2 discusses other distribution levels, together with their advantages and disadvantages and motivates the choice of decentralized event level distribution in this work.

Decentralized event level distribution operates as follows. The model is divided in the space domain and different parts of the space are assigned to different simulators (see figure 2.1). Each simulator executes the events in one part of the space (the model). A high degree of parallelism can be expected if there are only few dependencies between events in different subspaces. Figure 2.1 shows the space-time domain of one simulation. The points are events and the arrows represent dependencies. Event B depends on event A, D depends on B and so on. The event C is independent from A, B and D, so it can be executed in parallel to them. However, this level of distribution requires a *synchronisation protocol* to ensure the correct execution of the model. Otherwise, if both simulators have advanced their times up to t_1 and t_2 respectively, the event D affects event E in the past which is not correct.

Synchronisation There are two common approaches for synchronisation among LPs in decentralized event level distributed simulation. One of them is called *Conservative*, and the other one *Optimistic*. Their goal is to guarantee that a distributed simulation

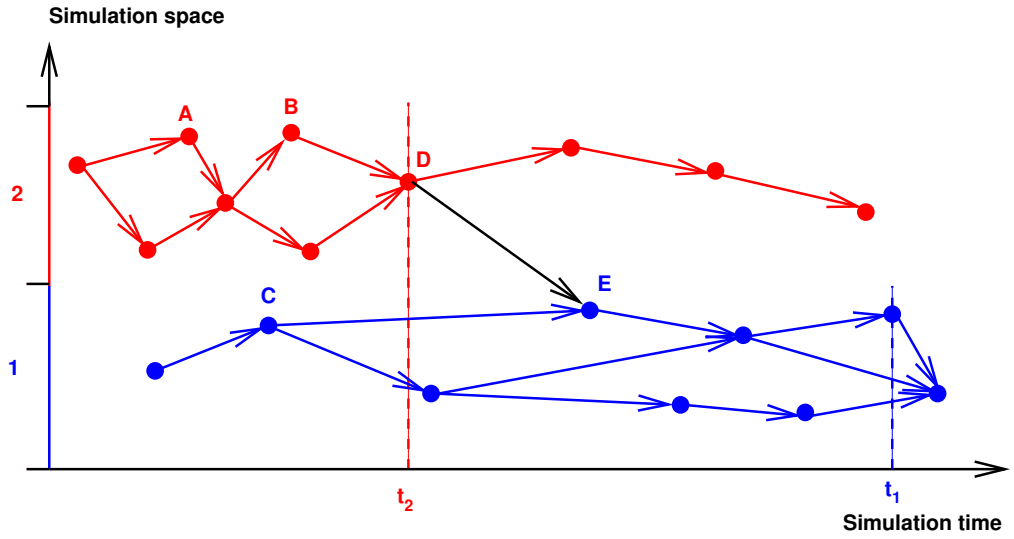


Figure 2.1: Space/time domain of event-level decentralized DES

produces *exactly the same* results as a sequential one with the same model. This does not necessary mean that all the events in the model have to be executed in timestamp order. It is only necessary that it seems so, i.e. independent events can be run in parallel or out of order. The conservative techniques keep the so called *local causality constraint*. This means that each LP guarantees that all the events it processes are strictly in timestamp order. This constraint avoids situations like the one depicted on figure 2.1 (event D affects event E in the past). The optimistic synchronisation protocols allow such situations, but they take measures to recover from them and guarantee the correctness of the simulation.

Conservative Synchronisation The conservative methods were first introduced by Chandy and Mirsa in [12]. They are based on the property *lookahead*. Lookahead is a time interval in the future in which an LP guarantees that it will not sent *any* messages to another LP. This interval starts with the current simulation time of the first LP and its length is defined by using model specific knowledge and may vary through the simulation. Each LP calculates its lookahead time interval and reports it to other LPs whom it may send a message. Based on this information each LP determines a so called *safe window*, in which it is sure that it will not become any message from any other LP. So all the events in this safe window are known to the LP and can be processed in timestamp order. This idea is illustrated on figure 2.2.

In this simulation there are two LPs. Both of them calculate their lookahead intervals at the current time T_1 and T_2. Each LP reports the upper bound of its lookahead interval to the other LP. The reported timestamp means that an LP will

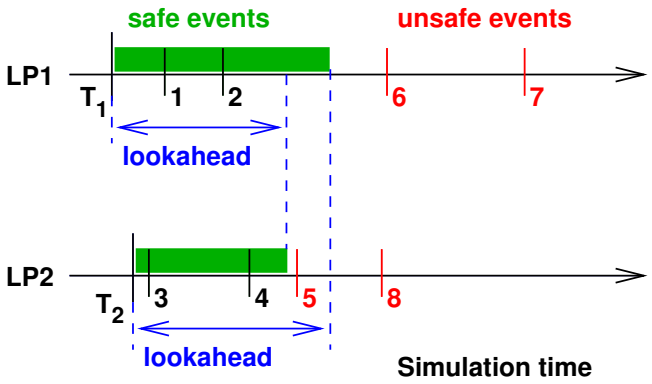


Figure 2.2: Conservative synchronisation

not send any message to the other one until that time. Based on this information and its current time each LP calculates its safe window where the events can be processed. The events after the safe window are not allowed for processing. The events {1, 2, 3, 4} on figure 2.2 are safe and can be processed in parallel, while the events {5, 6, 7, 8} are unsafe and can not be processed in parallel. If an LP reaches the end of its safe window it has to wait for the other LPs to reach that point and report their lookaheads.

Lets take a model of an Internet router as an example for extracting a lookahead. Let the minimum time it takes to forward one IP packet from an input port to an output port is T_f. Then, if the router does not have any packets in its buffers, it knows that there will be no packet sent through the output port at least for the next T_f. This is because even if a packet arrives immediately it will take at least T_f to forward it. The job processing time is a typical source of extracting lookahead from the model — it can be applied to other kinds of models.

The lookahead value has a direct influence on the speedup of a distributed simulation. If the lookahead is big, then the safe windows are also big and the LPs can operate in parallel with few synchronisations. If the lookahead value of a model is small, then the LPs will more often have to stop event processing and do synchronisation. This reduces the speedup, gained from a distributed execution.

Optimistic Synchronisation The optimistic strategies proceed with the execution of events as far as they can in the future and assume that no messages will come in the past. So, they allow violations in the local causality constraint but take measures to recover from them, if the above assumption is wrong. These measures are *state saving* and *rollback*. State saving is a backup of the state variables at a given simulation time. The backup contains all the necessary variables and structures that are needed to proceed with the model from that point in time. State saving can be done periodically, or after the execution of each event. It keeps a history of the system for the case that

the local causality constraint is violated. Rollback is a process of restoring a previously saved state. It is caused by a message, that has a time-stamp in the past simulation time i.e. *straggler message*. A straggler message indicates that the computations in the interval from its time-stamp to the current time are wrong and have to be thrown away. When a straggler message is received, the simulation is stopped and the most recent state before that message is restored. The message is then processed and the simulation is restarted again from that point in time. The restore of a previously saved state includes also the invalidation of all messages, sent during the wrong simulation interval. These messages are cancelled, by sending the so called *anti-messages*. An anti-message has the timestamp of the corresponding message and contains an indication that the previously send message is invalid. Anti-messages can also cause rollbacks, if they are received in the past simulation time. This phenomenon is called *rollback chains*, because a straggler message in one LP can cause rollbacks in many other LPs in the system. The most widely used optimistic synchronisation mechanism is Time Warp [23].

The optimistic techniques have two advantages over the conservative ones. First, they explore a higher level of the model's parallelism. If the events A and B occur periodically it may happen that A affects B not always, but occasionally depending on the model. A conservative method will always execute event A first, even in the cases in which it does not affect B. An optimistic approach will always try to execute the two events in parallel, and make use of every opportunity for parallelism. The second advantage of the optimistic methods is that they do not need model specific information such as lookahead. They can make use of it, but it is not required. So, an optimistic synchronisation can be more transparent to the model than a conservative one.

The optimistic technique has also disadvantages over the conservative one. The main disadvantage is that it is more complex. It requires implementation of state saving and rollbacks. The state saving requires a specific design of the model for distributed execution. The introduction of state saving mechanism in a sequential model is not a trivial task. The rollbacks require *"unsending"* of all messages, that are sent during the wrong interval of the simulation. This can cause chains of rollbacks, chasing down incorrect computations, and rollback echoes [18]. Care must be also taken for operations that can not be rolled back (I/O operations, program errors, possibly caused by the inconsistency of the model). Another disadvantage of the optimistic techniques is that they require more operating memory for state saving.

Time Difference *Time difference* between two LPs is the difference in their simulation times. This measure shows the difference in evolution of the simulators. Generally speaking, a small time difference among LPs in one simulation is a good property. It is valid for both conservative and optimistic synchronisation. This is

because a conservative simulation develops with the speed of the slowest LP. The faster LPs have to wait for its allowance before they can proceed. So, the smaller the time difference, the smaller are the waiting periods, and the higher is the utilisation. In an optimistic simulation each LP develops with its own speed. But the closer the LPs are, the fewer computations have to be thrown away if a straggler message arrives in the past. If the LPs are closer the rollbacks are also simpler, because the probability for anti-messages and chains of rollbacks decreases.

2.3 Wireless IEEE 802.11 Networks

IEEE 802.11 is a standard for Wireless LAN Medium Access (MAC) and Physical Layer (PHY) Specifications. It specifies an over-the-air protocol for local area network communication [22, 19, 13]. The term *wireless* means that these networks do not use cables (wires) for transmission of signals. The wireless networks use other physical phenomena that allow to transmit signals through the air.

There are primary two types of wireless networks, depending on their physical organisation: *infrastructure* and *ad-hoc*. Infrastructure networks use an existing and structured base network. It consists of *access points*, which are fixed stations and may provide connectivity to other networks. Mobile stations are assigned to the access points and can use them to connect to other parts of the network. On the other hand, ad-hoc networks do not use any predefined network structure. They consist solely of stations within a mutual communication range with each other and communicate via the wireless medium[22]. These networks are created in a spontaneous manner without the need of specific technical skills. An extended variant of ad-hoc networks are the *multi-hop ad-hoc networks*. They spread over larger areas and it is no longer possible for every two stations to communicate directly through the medium. These networks use dynamic routing protocols, where each station, besides sending and receiving own packets, serves as a router to other stations. The stations that are out of communication range search for a path through the network and communicate via other stations.

The main parts of the IEEE 802.11 standard are the Medium Access Control (MAC) and Physical (PHY) layer specifications for both infrastructure and ad-hoc networks. The MAC layer is a part of the Data Link layer in the OSI Reference Model. It controls the access to the physical medium, detects transmission errors at a frame level. Additionally it transforms frames into sequence of bits and reverse, and notifies the upper layers when a frame is destined for the host. The physical (PHY) layer is the lowest layer in the OSI Model. The sender PHY encodes the sequence of bits from the MAC layer in a transmission signal. The receiver PHY decodes the sequence of bits from the signal, and possibly detects and corrects transmission errors at a bit-stream level. This layer specifies physical parameters of the medium like frequency of the signal and transmission power.

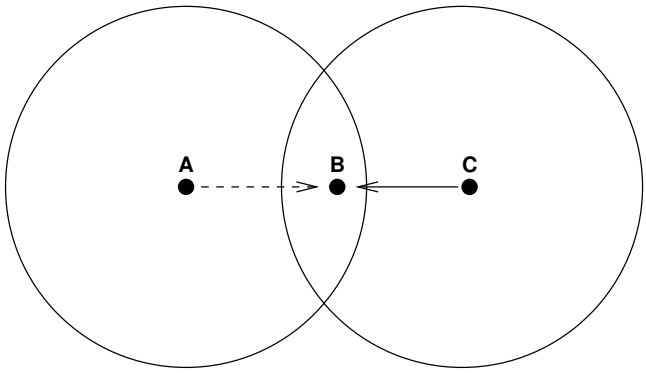


Figure 2.3: Hidden station problem

IEEE 802.11 MAC

The WLAN standard 802.11 provides two methods for medium access — *Point Coordination Function* (PCF) and *Distributed Coordination Function* (DCF).

PCF The Point Coordination Function is a priority-based access to the medium, that can be used in infrastructure networks. A central controller rules the medium access in the network. The controller polls the stations sequentially in a logical ring, and each poll is a permission (grant) to send one frame to the medium. This medium access method is contention-free and can provide deterministic packet transmission times. Therefore, it is used for real-time communications.

DCF The DCF uses a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) method. It is a contention-based medium access method, similar to the Ethernet (CSMA/CD). CSMA/CA is used in ad-hoc networks, since there is no central controller there. The DCF uses two types of carrier sense — *physical carrier sense* and *virtual carrier sense*. The first one is achieved by using the status of the physical interface. It reports whether the physical medium is *busy* or *idle* — similarly to the Ethernet carrier sense. Physical carrier sense is sufficient in Ethernet, because all the stations share the same physical bus and every station can sense a transmission from any other. However, it does not suffice to detect collisions in wireless networks because of fading of the signal. Therefore physical carrier sense provides only local information and can not determine whether the medium is busy or idle at the receiver side. This can be illustrated by the popular "hidden station problem"[31]. It is depicted on figure 2.3.

The figure shows three wireless nodes — A, B and C. The circles around them show the range in which other nodes can sense their transmission signals. Node A can sense a transmission from node B, but not from node C because it is too far away. Node B

is in the middle and can sense transmissions from both A and C. Now imagine that node C is transmitting a frame to node B and at this time node A decides also to transmit to node B. If A used only its physical carries sense it would not sense the transmission from node C. So it would decide that the medium is free and transmit. But both transmissions would collide at the receiver B and neither of the frames would be received correctly.

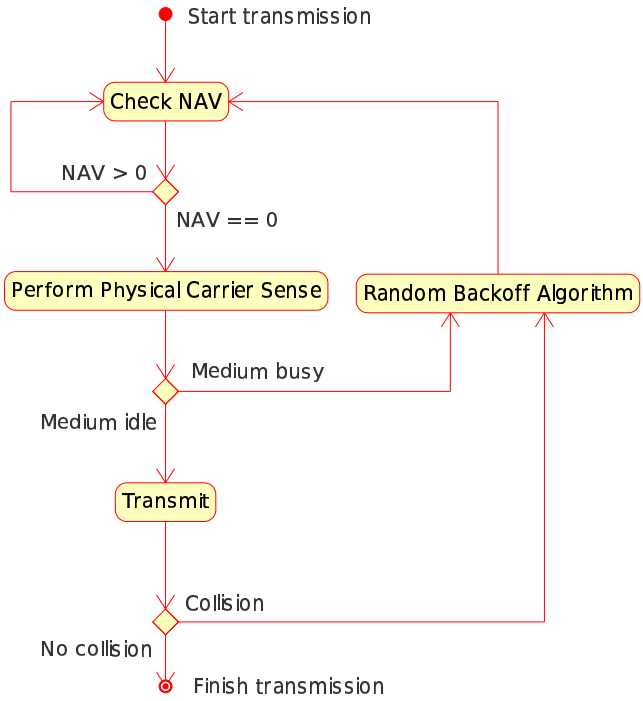
Virtual Carrier Sense To reduce the probability that this problem occurs the standard uses virtual carrier sense. It is achieved by the MAC layer by placing duration information in each frame. It shows the duration of a current transmission and informs other stations to consider it in their medium access. The stations use this information to maintain a *Network Allocation Vector* (NAV). This is a period of time in which the medium is busy. The virtual carrier sense mechanism is implemented with RTS (Request To Send) and CTS (Clear To Send) control frames. These are short messages that inform other nodes in the network that a transmission will take place. When a node wants to transmit a data frame it first sends an RTS frame to check whether the medium at the receiver side is free. If the medium is free the receiver replies with a CTS frame and then the transmission can start. The RTS and CTS frames contain the duration of the intended transmission. All other nodes within a transmission range are supposed to receive either the RTS, or the CTS (or both) and update their NAV timers according the duration of the transmission[1]. So, the NAV shows whether a station is allowed to transmit or not. After the RTS/CTS handshaking the sender transmits the data frame. If the destination receives the data correctly it replies with an acknowledgement frame.

Inter-frame spaces The DCF defines priorities of the different frame types by using *Inter-frame spaces*. Inter-frame space (IFS) is a minimum time which should pass between the transmission of two frames on the medium. There are three types if inter-frame spaces used in DCF. They are in the range of $[2...50\mu s]$.

DCF Algorithm The basic algorithm of Distributed Coordination Function is described by algorithm 2.2. The station first checks virtual carrier sense, then the physical carrier sense and transmits if both of them allow. If the physical carrier sense indicates that the medium is busy, or a collision occurs the station uses a random backoff algorithm to reduce the probability of a subsequent collision[13].

[1]The signal in wireless ad-hoc networks propagates equally in all directions from the sender. Therefore the transmission area is a circle with centre — the sending node, and radius — the transmission range.

Algorithm 2.2 Algorithm of IEEE 802.11 MAC DCF



IEEE 802.11 PHY

The physical layer of the WLAN standard IEEE 802.11 provides three methods for wireless transmissions. Two of them use radio waves in the licence-free ISM frequency band 2.4GHz and the other one uses infra-red (IR) light with wavelength 850 to 900 nm. The radio methods use spread-spectrum techniques to divide the wireless medium into set of communication channels. This allows the coexistence of multiple independent networks within one area. Due to the popularity of radio networks in the recent years the term "wireless" is widely used to denote radio networks. This is also the meaning of "wireless" in the title of this thesis.

PHY Operations The main functions of the physical layer are *Carrier sense*, *Transmit* and *Receive*. Carrier sense is used for detection of incoming signals, and for clear channel assessment. The second reports to the MAC layer whether the medium is *busy* or *idle*. The transmit function sends a frame in the network. It is used by the MAC layer when it has a frame to transmit. The receive function receives a frame from the wireless channel, and forwards it up to the MAC layer.

Radio Signalling The properties of the radio signal are important to understand the operation of wireless networks. Key concepts are *interference*, *transmission range* and *carrier-sense threshold*.

- Radio Interference

  Radio interference is the influence of other radio waves in the network on a given radio signal. If this influence is low enough, the signal can still be received correctly. If it is too high it causes the incorrect reception of the signal. The definitions of "low enough" and "too high" depend on the used modulation scheme, error detection and correction mechanisms.

  The IEEE 802.11 standard uses a measure, called *signal-to-interference-plus-noise ratio* (SINR) to determine the quality of a signal. SINR is the ratio of the reception power of the signal of interest P_{sig} and other signals at the receiver. Other signals are all interference signals with power P_{intf} and the noise with power N.

  $$SINR = \frac{P_{sig}}{\sum P_{intf} + N}$$

  Each radio receiver defines a minimal $SINR$ which is required for the correct reception of a signal ($SINR > SINR_{min}$). The value of $SINR_{min}$depends on the error-coding and modulation methods used.

- Transmission Range

  $SINR_{min}$ determines the *transmission range* of a transmitter and a corresponding receiver. It is the maximum distance at which the transmitted signal can be received correctly in the absence of any other interference signals. The transmission range is usually equal for every two stations in the network and it is also called *radio range*.

- Carrier-sense Threshold

  A station in a radio network determines whether the medium is busy or idle based on a preconfigured carrier-sense threshold (T_{CS}). This is the minimum power of a carrier signal that is relevant to a station. If the power of the carrier is smaller than T_{CS} the medium is considered as idle. *Carrier sensing range* is the distance within which all transmissions have at least a power of T_{CS}. Typically T_{CS} is chosen such that the carrier sensing range is at least as the transmission range.

2.4 The ns-2 Simulator

Ns-2 (Network Simulator 2) is a discrete-event network simulator[9]. Its input is a description of a network model, and its output is an imaginary history of this network. The network model consists of *network topology* and *traffic patterns*. The network topology describes the structure of the network together with activities during its operation. Structural information is for instance the coordinates of nodes, the type of physical network and networking protocols at different network layers. Activity may be for instance a failure of a network node, an increase of packet transmission errors due to interference with a neighbouring network, and many others. The traffic patterns describe the behaviour of the applications in the network. This description is application specific, but it usually defines the rate and size of packets which an application sends in the network. These parameters can also have a random distribution.

Ns-2 has mainly two kinds of output — a *trace file* and an *animation file*. The trace file contains information about the history of the network, and it is intended for statistical analysis. It includes an entry for each packet that is send in the network between two nodes, or between two network layers in the same node. The entry contains simulation time, source and destination nodes, packet sequence number and other information needed for numerical evaluation of the network. The animation file contains animation commands, that can be used together with the animation tool nam (Network Animator)[6]. It can visually represent the simulated network via an animation.

Ns-2 is a sequential simulator — it uses the standard discrete-event-simulator algorithm, described in section 2.1. The main part of this algorithm is the simulator

scheduler. This is a controller that maintains the event list, ordered by the timestamps of the events. The scheduler takes the next earliest event from the list and executes it. The event performs an activity and possibly *schedules* (inserts) another events into the scheduler list. After the execution the control comes again into the scheduler to execute the next event.

Radio Model The ns-2 simulator implements a basic part of the IEEE 802.11 standard[22]. The MAC layer realises the Distributed Coordination Function (DCF) by a CSMA/CA medium access method. Physical carrier sense is provided by the physical layer and virtual carrier sense is implemented via RTS/CTS frames and the use of a Network Allocation Vector (NAV). The data frames are transmitted using the four-step frame-exchange sequence RTS / CTS / Data / Acknowledgement. The MAC layer in ns-2 accounts for collisions, processes packets with transmission errors and uses the inter-frame spaces of the standard (IFS). The random backoff algorithm is also implemented.

The physical layer provides an abstraction of a wireless channel. It provides the basic functions: Carrier Sense, Transmit and Receive, and implements a basic model of radio signalling. The physical layer in ns-2 uses the same transmission power for all wireless nodes. It calculates the propagation of the signal using radio-propagation models: the free space model[17] and the two-ray ground model[29]. The power of the signal at the receiver determines whether a frame can be received or not. The physical layer uses two predefined constants *Receive Threshold* ($RXThresh$) and *Carrier-sense Threshold* ($CSThresh$) to compare the power of an incoming signal P_{sig}. Depending on the value of P_{sig} three cases are possible:

1. $P_{sig} \geq RXThresh$

   In this case the signal can be received correctly. Then it keeps the medium *busy* for the duration of the transmission. The frame is delivered to the MAC layer which can receive it, or discard it if a collision has occurred. If the MAC layer receives the frame correctly it updates its Network Allocation Vector. Then if the frame is destined for the node, the MAC layer forwards it to the upper layers.

2. $RXThresh > P_{sig} \geq CSThresh$

   In this case the signal can be received, but with transmission errors. The signal keeps the busy state of the medium, and the PHY notifies the MAC for an incorrect reception of a frame.

3. $CSThresh > P_{sig}$

   In this case the signal is ignored by the physical layer. This is a simplification at the physical layer, because in reality the signal would contribute to the $SINR$ at

the receiver. However ns-2 does not accumulate the incoming signals to calculate $SINR$, but uses the constants $RXThresh$ and $CSThresh$ for this purpose. So this signal is irrelevant to the physical model in ns-2.

Distributed Simulation The carrier-sense threshold $CSThresh$ determines the value of a variable, relevant to distributed simulation — $distCST$ (Distance Carrier-sense Threshold). This is the maximum distance at which the signal from a transmitting station has at least power of $CSThresh$. In other words $distCST$ is the maximum distance at which other nodes in the network should be notified when a particular node transmits. This property of the model can be used in distributed simulation because it defines a range in which one node can affect other nodes. This range is also called *sensing range*.

I assume that the nodes in ns-2 communicate *only* via frame exchange through the wireless medium. This means that there are no direct calls of methods of one node from another node. This assumption can hardly be proved due to the complexity of the model, but it is very likely to be true because it is required for a correct simulation design. Otherwise, a network in which the nodes can call methods or exchange messages directly and not through the medium would not make sense. Based on this assumption and the sensing range, the following statements can be used to distribute the model of ns-2:

- If two nodes lie at a distance more than $distCST$ (out of sensing range), they *do not affect* each other and can be simulated in parallel without synchronisation.
- If two nodes lie within a sensing range they *might affect* each other and their actions have to be synchronised.

2.5 Lookahead in Wireless Network Models

The simulation models, used in the context of this thesis are models of wireless ad-hoc networks. They are based on the IEEE 802.11 standard using the Distributed Coordination Function of the Data Link layer. In order to simulate these networks in a distributed way the models have to be studied for the parallelism available in them. This is essentially lookahead extraction. There are two types of lookaheads, that can be extracted from these models: *static* and *dynamic*.

Static Lookahead Static lookahead is always constant, and does not depend on the status of the simulation model. This kind of lookahead is usually determined by the minimum possible time for a physical phenomenon. For instance, if a simulation is modelling airports, a static lookahead can be the minimum time for an airplane to fly to another airport and affect the model there. A similar phenomenon can be used in

computer networks. A static lookahead here is the time for a packet to "fly" from one subnetwork to another. But since a packet affects another subnetwork starting from its *first* byte, the lookahead is the time for this byte to travel through the network. This is the *network propagation delay*. Signal propagation in wireless networks is the propagation of radio waves, and therefore it has the speed of light. Since the distances in ad-hoc wireless networks are very small (maximum 550m in the model of ns-2), this propagation delay is also very small (1...1500ns). Network simulations with ns-2 run in a magnitude of minutes of simulation time, and therefore this static lookahead from propagation delay is very small.

Dynamic Lookahead Other sources of lookahead may have a dynamical nature. These lookaheads depend on a current status of the model, and may change over time. For instance, in the airports model, if the current weather conditions do not allow an airplane to take off, it will arrive later at the destination and give a possibility for a higher lookahead. These dynamic lookaheads require a synchronisation protocol, which is aware that lookahead may increase and decrease during the simulation. In the case of wireless networks, lookaheads can be extracted from the operation of the Medium Access Control (MAC) layer, since it controls sending of packets and affecting other parts of the network. The algorithm of the Distributed Coordination Function (DCF) can be used for this purpose (see algorithm 2.2).

This algorithm shows that if another station is currently transmitting (NAV > 0) the current station will not send to the medium. This allows to use the value of the NAV as a lookahead. Since wireless transmissions take time in a magnitude of milliseconds, the NAV can promise lookaheads of this range (usually $[1...2ms]$, depending on the packet size and the transmission rate). But if the medium is free a station transmits immediately. In this case a source of lookahead are the inter-frame spaces. Their length is in the microseconds range $[2...50\mu s]$, i.e. much smaller than lookahead, extracted from the NAV.

The lookahead of one simulator depends on the lookaheads, extracted from all network nodes, that it is simulating. Even if only one network node reports a free medium and a lookahead in the range $[2...50\mu s]$ this small lookahead has to be considered as a lookahead of the simulator. Since one simulator models many network nodes, the probability that one of them provides a short lookahead is high. The study [24] investigates lookaheads in wireless networks, and concludes that optimisation techniques can increase the lookahead to $[5...60\mu s]$ in 90% of the lookahead computations. This lookahead is far too small to achieve speedup in distributed simulations. Therefore I am considering also upper network layers, specifically in the context of this thesis, in order to achieve higher lookaheads.

These upper layers are the communication protocols, that are developed and tested with ns-2 in the context of this thesis. The goal of these protocols is to guarantee

timely delivery of packets in multi-hop wireless ad-hoc networks. But since the DCF mechanism used there is contention-based, it does not give predictable transmission times, and is not appropriate for real-time communications. Therefore, these upper layers build an access mechanism similar to the Point Coordination Function (PCF) over the DCF. In these scenario some network nodes take the role of a central coordinator (*cluster head*), and other nodes take the role of a *client station*. The cluster heads continuously poll the clients in order to give them access to the medium. A cluster head sends a poll once per time interval, which is fixed in the range of $[20...30ms]$. A client sends a packet, only when it receives an invitation from a cluster head. This means that if the medium is free at the client side it can not send immediately, but after receiving an invitation, which takes time at least for the transmission of one data frame (usually $[1..2ms]$).

So, network models in the context of the thesis have larger lookaheads than standard wireless networks. They can be in the range of $[1...30ms]$, and their behaviour can be determined by an additional study. These lookaheads can make the models appropriate for conservative distributed simulation. However, the lookaheads have a dynamic nature, and require appropriate methods for dynamic lookahead extraction and use.

3 Related Work Study

This chapter discusses other works and efforts in the area of parallel/distributed simulation of wireless ad-hoc networks. First it reveals the purposes of this related work study. Then it discusses research projects, which address problems, similar to this thesis. Finally it summarises ideas from the related work study that are helpful to this thesis.

3.1 Purposes of the Study

One purpose of the related work study is to find other research projects, similar to this thesis, that may contribute to it. The contribution is knowledge and experience gained from these projects that can be helpful to the thesis. The contributions may have a *positive* nature, i.e. an idea from another project that can be applied to the thesis. But they may also have a *negative* nature, i.e. an idea or a method, used in another project, that is not applicable in the context of this thesis.

Another purpose of the related work study is to obtain a base for a comparison of the solution of the thesis with other similar efforts. This helps to evaluate the result of the thesis.

3.2 Ideas from Other Research Projects

PDNS The Parallel and Distributed ns-2 (PDNS) is an extension to the ns-2 simulator for parallel and distributed simulation of *wired* networks[7]. It divides a model of a network into models of subnetworks and assigns each subnet to a separate logical process (LP). The LPs use a conservative (blocking) approach for synchronisation, i.e. each LP executes only *safe* events and does not violate the local causality constraint. This avoids the implementation of state saving in the existing ns-2 code. PDNS introduces the notion of *remote link* in ns-2. This is a network link that connects two subnets on different LPs. PDNS uses remote links to extract lookahead from the model. The propagation delay of a network link is the time for one bit to "travel" along the link. Therefore the propagation delay of a remote link defines the minimum simulation time that has to pass, before a packet, sent from one side of the link, affects the subnetwork on the other side. So the lookahead of an LP is the minimum network propagation delay of all its outgoing remote links. PDNS can be used together with a tool for automatic partition of the input network model — Autopart[1]. Autopart transforms a description of a network model for ns-2 into a description of a network model for PDNS. It uses graph partitioning algorithms[25, 26] to partition the input network model. Autopart eliminates the need of partitioning by hand and optimises the partition for best performance of PDNS. It is especially needed and useful for large-scale network simulations.

Optimisation Parallel and Distributed simulations can usually be optimised by using model specific knowledge. Ji et. al. have shown in [24] some properties of wireless networks which can optimise the parallel/distributed execution of their models. They discuss properties that help to increase the lookahead of a model in parallel and distributed simulations. Lookahead can be extracted at the MAC layer by inspecting the network allocation vector, the current inter-frame-space time, the value of the backoff timer. All these values show a minimum time period in the future in which a node will not transmit. Another source of lookahead is the duration of a current incoming frame. Since the radio antennas are half-duplex i.e. can not transmit and receive concurrently, a node will not transmit until it finishes the reception of a frame.

Ji et. al. use the network simulator GloMoSim[34] and explore its features to extract lookahead at the physical layer. GloMoSim accumulates radio transmission signals at the physical layer to calculate the signal-to-interference-plus-noise ratio. Ji et. al. use the accumulated signals to calculate the earliest possible moment in future time when a node *can* sense the medium as idle. Since idle medium is a requirement for a transmission this time can also be used to compute lookahead.

Ji et. al. use parallel simulation and achieve a speedup of 5.8 using 1000 wireless nodes on a 16-processor machine.

Statistical Accumulation Madnani and Szymanski present another method for optimisation of distributed simulation of wireless networks[28]. They divide a network model into geographical *domains* with rectangular shape. Each domain, together with the nodes inside is assigned to a separate LP. Each LP computes its *domain closure*, which is the area around its domain interesting for it. The domain closure contains other nodes that might affect the nodes in the domain of an LP. The simulation runs synchronously and continuously changes between two phases: *simulation* and *information exchange*. During the simulation phase each LP simulates its domain up to a specified simulation time. Then all LPs stop simulation and exchange information about activities, happened in the completed phase. During simulation, each LP simulates *in detail* the nodes in its own domain and *statistically estimates* the activities of nodes in other domains, i.e. in its domain closure. These activities include packets, send or received by the nodes in the domain closure. In the first iteration the LPs assume no activities in the domain closure. In the next iterations the LPs estimate the activities in the domain closure by the activities in previous iteration steps, acquired during the information exchange phases. Of course this method does not guarantee the same simulation results as a sequential simulation. It produces an estimation of these results with an error that depends on the simulation model.

This method achieves a speedup of 14.7 with a model containing 900 simulation nodes, executed on an IBM Netfinity cluster of 16 processors. The results from the distributed simulation differ by about 4.3% from the results from the sequential one.

These results are the values of interest, measured from the simulation model, i.e. throughput, packet delays, packet drops.

Adaptive Simulation Bononi et al. present in [11] a technique for adaptive optimisation of communication costs and load balancing in distributed simulation of mobile ad-hoc networks. They use a federate approach where a group of wireless nodes is assigned to each LP. The nodes exchange messages during simulation, which are either *internal* (within the same LP), or *external* (to other LPs). Bononi et al. use a simple migration policy of mobile nodes to reduce communication overhead among the LPs. The simulation algorithm migrates nodes during simulation from one LP to another trying to minimise the amount of communications among the LPs. It computes the ratio of external to internal messages for each mobile node, related to each foreign LP. The LP with a maximum ratio is a candidate for the next "home" of a mobile node. The migration process considers also the load balancing among simulators. It tries to keep the nodes in the simulated network uniformly distributed among the LPs.

In preliminary test simulation runs this method achieves a speedup of 1.5 with a model of 5000 nodes running on 3 computers, connected by a Fast Ethernet network.

3.3 Contributions to the Thesis

PDNS The PDNS project shows that a parallel/distributed execution of the network simulator ns-2 is possible. Special model-related changes are required to keep the integrity of the model across the simulators. The idea for model partitioning using a graph can be also adopted in this thesis. PDNS has also a technical contribution — it is a ready implementation of a distributed ns-2.

However, the method used for lookahead extraction is not very promising for this thesis. Wired links may cover long distances (e.g. international or transcontinental links) and may have a relatively high propagation delays. Wireless ad-hoc networks on the other hand cover relatively small areas and therefore have much smaller propagation delays.

Optimisation Ji. et. al. show general properties of wireless networks at the data link layer, that can be also used for lookahead extraction in this thesis. The suggested properties of the physical layer model can not be used in this thesis. This is because ns-2 has a less detailed simulation of the physical layer than GloMoSim, e.g. it does not simulate accumulation of wireless signals.

Statistical Accumulation The method of statistical estimation of neighbouring nodes is interesting for this thesis, because it gives a good speedup and a good approximation of a sequential simulation. However, the quality of the approximation

strongly depends on the simulation model and the points of interest in it. If the interest is a measure of average performance statistics of the network, this method might be appropriate. However, the goal of simulations in the context of this thesis is to test the behaviour of a distributed application or a protocol in a wireless network. So, this method is less appropriate, because it introduces an artificial, statistically estimated data into the network. In this way it breaks the semantics of the communication among applications. For this reason, this method is not chosen in the thesis.

Adaptive Simulation The usage of the method for adaptive repartition and load balancing depends on the used simulator. If the migration of mobile nodes from one simulator to another one is a straightforward process, this method is applicable. Bononi et al. represent mobile nodes by structures that store all state variables and can easily be serialised and transfered to another simulation host. However, wireless nodes in ns-2 are designed for a sequential simulation and consist of multiple objects, linked to the simulator engine by a series of pointers. This makes the serialisation of a wireless node in ns-2 a non trivial task, and therefore the method of adaptive node migration is not chosen in this thesis.

4 Design of the Simulation System

This section explains the design of the thesis on an abstract level. It discusses ideas for a solution and gives reasons for the taken design decisions. Section 4.1 describes the intentions of the design and the criteria for taking design decisions. Section 4.2 motivates a fundamental design decision for choice of a type of distribution in the distributed simulation system. After that, section 4.3 gives an abstract, overall description of the chosen system design. Section 4.4 gives details about the components of the distributed simulation system. It describes partitioning and distributed execution of the simulation model. Section 4.5 describes a design idea to ease the distribution of the simulation model. Finally section 4.6 estimates the contributions of the design and its consequences for the thesis.

4.1 Design Purposes

The purpose of the design is to choose an overall structure of a distributed simulation system and its components. At this step I will choose one of the possible ways to solve the thesis task. The chosen method will try to satisfy the following criteria:

1. Optimal for this task and its specific requirements. The solution should use task specific requirements, and information in the context of the thesis in order to maximise the significance of the results. The thesis does not try to solve the problem of distributed simulation of wireless networks in a general case. It is concentrated on specific wireless networks (see section 2.5 for details about the context of the thesis).

2. Straightforward technical implementation. Because of the relatively high complexity of the task, I prefer straightforward methods than more complicated ones. This aims to keep the task feasible and avoid an additional increase of its complexity.

3. Independence of different system components. Different blocks of the system should rely only on interfaces between them and should not depend on the operation of other blocks. This gives a modular design and allows to change modules of the system transparently from other modules.

Of course it might not be possible to optimise the solution in all directions, because they might have a contrary nature. For example, the optimal solution does not always have a straightforward implementation. I am going to choose a tradeoff in such cases.

4.2 Choice of Distribution Type

The first decision by the design of a distributed simulation system is to choose the type (level) of distribution. This is a fundamental decision, that determines the design of the system, and therefore it is taken first. Section 2.2 already described the decentralized event level distribution. Here, I discuss several other distribution levels [15], and give reasons for the choice in this work.

1. Application level

   This is the most intuitive and trivial way to distribute a DES if the model contains random variables. The sequential simulator is run on different computers with the same model, but different sequences of random numbers. This method can be used when independent replications are needed to obtain statistically significant simulation results [14]. It is the most easiest and fastest way to distribute a simulation. First, because no changes in the simulator are needed and second, the different runs are absolutely independent from each other and require no synchronisation. However, the goal of this work is to speedup a single simulation run and this trivial distribution is not appropriate.

2. Subroutine level

   At this level, simulation subroutines, that have supporting function, are distributed to other computers. These include random number generation or collection of statistics. Other supporting tasks appropriate in multiprocessor environments are memory management or postprocessing of simulation results. Due to the relatively small number of possible support subroutines in ns-2, this level of distribution does not promise a significant speedup.

3. Component level

   This distribution is on the software modules level. Different modules of a simulation program are executed on different computers. In network simulation this would mean that the model of each layer is executed on a separate simulator. This approach is not appropriate in this work, because there are not many network layers or other components that can be separated — it has a limited scalability. Moreover, network layers exchange messages frequently which would lead to a significant communication overhead among the simulators.

4. Event level - centralized

   This level distributes the execution of single events to other computers. The method is centralized — a master processor maintains the event list. It assigns heavily weighted events with the same timestamp or close to each other to be executed by the slave processors. This method makes use of independence

between events in the time domain, i.e. it uses concurrency of events to achieve speedup. So, this method makes less use of independent events, than the decentralized method, which also uses the space domain. Another motive against this method is that it does not have a straightforward implementation in ns-2. Ns-2 is by design a sequential simulator and does not provide primitives to execute events on another hosts.

5. Event level - decentralised

   This level divides the simulation model in space and time, and assigns each portion to a separate processor. It promises a high speedup if the model has many independent events in these two domains. However, this method requires a synchronisation protocol to ensure the integrity of the simulation model. This method of distribution was discussed in detail in section 2.2.

The decentralized event level distribution makes use of independent events in both space and time domain, so it explores a better degree of the model's parallelism than the other levels. Therefore it promises a greater speedup. Its disadvantage is that it is more complex than the others and requires distributed synchronisation protocols. However, a lot of research and development is done in this field, and I rely to use efforts and achievements of other researchers in this work. For these reasons I choose the decentralized event level distribution in this thesis work.

4.3 Overall Design

After the level of distribution is fixed, the next step is to choose an overall design of the distributed simulation system. Since this thesis task is an extension to the network simulator ns-2, the overall system design should be similar to it. The input is a description of a network model, like in ns-2, and the output is also a history of the simulated network. The solution differs from ns-2 by the distributed execution of the simulation model. The definition of the thesis task requires at least two basic blocks in the system — modules for automatic partitioning and for distributed execution of the model. The chosen level of distribution implies two other blocks for synchronisation and exchange of messages among logical processes. An additional module for postprocessing of simulation results completes the overall design of the system. Figure 4.1 illustrates this design.

The figure shows the blocks of the system, together with interactions between them. The input to the system, and its output are the same as the input and output of ns-2. The system has some additional blocks, needed to accomplish the task for distributed simulation. The block for model partitioning divides a sequential ns-2 model into multiple partial models. Then the component for distributed execution runs the partial models in parallel. This component uses the "Synchronisation" and "Message Exchange"

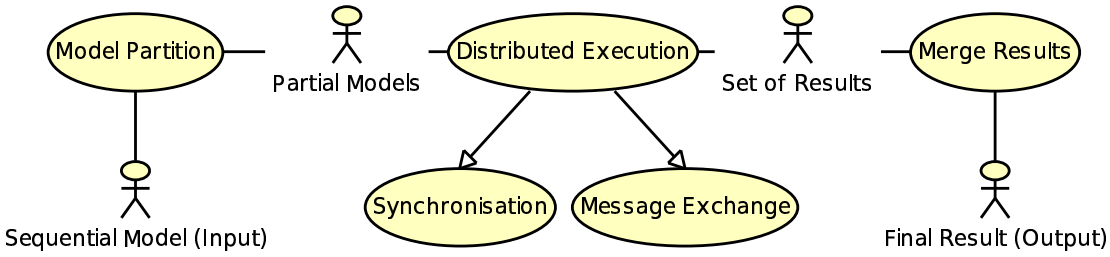


Figure 4.1: Overall system design

modules to ensure the correct execution of the model and its consistence. The module for distributed execution, together with "Synchronisation" and "Message Exchange" is the main part of the system. The product of the distributed execution is a set of simulation results, corresponding to the different parts of the model. These results are then merged to compose the final simulation result. This final result should be the same as a result from ns-2 with the same input model, but it should be generated faster.

4.4 Design of System Components

Following the overall design, this section gives details about the system components and motivates their choice. It is concentrated on three components: "Model Partition", "Synchronisation", and "Message Exchange". The component "Distributed Execution" has a general function to define the level of distribution, which was already discussed in a previous section 4.2. It mainly consists of "Synchronisation" and "Message Exchange". The component "Merge Results" has a straightforward operation. It simply combines and orders results from sub-models to form a history of a network, corresponding to the whole input model.

4.4.1 Method for Model Partitioning

The goals of the partitioning are:

1. Divide the model of the network into *roughly equal* parts (spaces). These parts are mapped to logical processes. Roughly equal means that the needed processing time to simulate a network part is close to the time, needed for another network part.

2. Minimise the exchange of synchronisation messages among the logical processes.

Both goals are significant for speedup in distributed simulation. The first one aims at a good balance of the model among the simulators. This means that the LPs will proceed in time close to each other and have a small time difference. This is considered a good characteristic of a distributed DES for both conservative and optimistic synchronisation (see section 2.2). The second goal tries to achieve a lower synchronisation among the logical processes. It means that they can operate independently for longer periods, which increases the speedup, gained from distributed simulation.

The first step in the design of a component for topology partitioning is to choose the unit of the partitioning. It is a structure that remains as a whole after the partitioning. The instances (objects) of this structure can be assigned to different LPs to partition the network. A reasonable decision by network partitioning is to choose a *network node* as a partitioning unit. This is because a it is a relatively closed system. Many events happen inside a node: exchange of packets among the network layers, scheduling of timers, changes of the states of the network layers. On the other hand, relatively fewer events happen outside a network node. These are primary send and receive packets to/from the network. So, if the node is the partitioning unit, the messages, exchanged among the LPs are the relatively rare compared to the events inside the LPs. There is another reason to choose the network node as a partitioning unit. If each node in the network requires an amount of processing time for its simulation, the load of each LP can be estimated by the sum of processing times of all nodes in it. So, the partitioning unit "node" contributes to both goals of the partitioning, and therefore I choose to partition the network on the network nodes.

The next step in the design is to choose a method to partition the network into groups of nodes. One idea is to split the network geographically using some geometrical forms. Then the group of nodes that lie in each area is assigned to a logical process and simulated separately. This method is easy to implement when the geometrical forms are homogeneous. It is showed on figure 4.2a. The figure shows the area of a wireless network, where the points are nodes. However, such separation might not consider the structure of the network and have a low quality. The upper left corner contains too many nodes, and the lower left too few. In order to consider the structure of the network one might need different types of geometrical forms and of different size, as it is shown on figure 4.2b. These geometrical forms consider the structure of the network, but may increase the complexity of the solution.

Another idea is to represent the network as a graph and then partition it using a graph partitioning algorithm. This method is intuitive, because a network can be easily represented as a graph. The vertices correspond to the network nodes and the edges correspond to the links among the nodes. The vertices may have a weight, representing the processing time, that a network node needs for its simulation. The edges may also have weights, which represent the amount of communications between two network nodes. So, this method considers the structure of the network. A graph

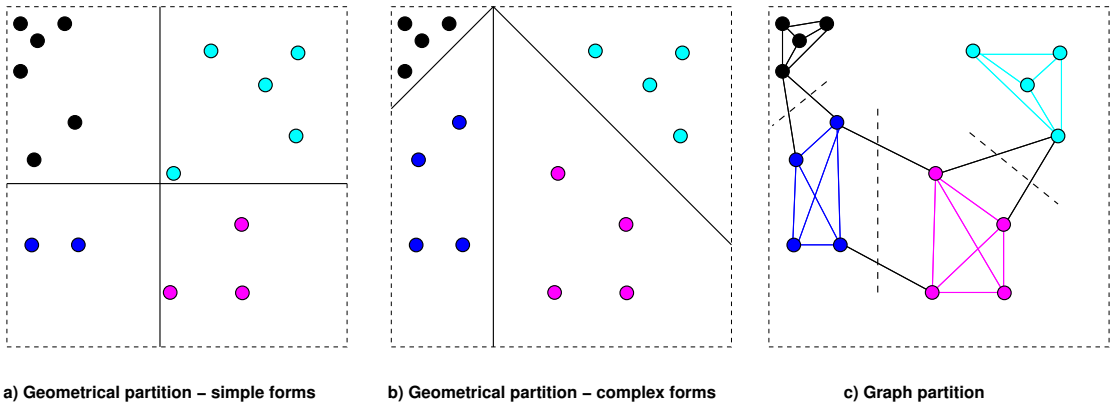


Figure 4.2: Network partitioning methods

partitioning algorithm can divide the graph into roughly equal parts and minimise the sum of communications between partitions. So this method contributes to both goals of graph partitioning, defined above. It is also relatively easy to implement because the problem of graph partitioning is well known in the graph theory [25, 26]. There are also studies that investigate graph partitioning algorithms for distributed simulations [27].

So, the chosen method for network partitioning is the following. The network is transformed to a graph, then the graph is partitioned and the partitioning is mapped to the network. Each partition of the graph (network) is assigned to a separate logical process.

The transformation from network to graph includes generation of vertices and edges of the graph from the network topology. The vertices of the graph are the nodes in the network. Since wireless networks do not have links, another property is used to create the edges of the graph. Wireless networks have a broadcast nature, i.e. a packet on the physical layer is sent to all the nodes in a *sensing range*. So, the sensing range of the nodes is used to define logical links in the network and edges in the graph. This idea is illustrated on figure 4.2c. It shows a graph that represents the wireless network on figures 4.2 *a* and *b*. Two vertices are connected by an edge if the corresponding nodes are within a sensing range, and can affect each other in a simulation. The edges that are "cut" by the partition are marked by a dashed line.

4.4.2 Method for Synchronisation

After the network topology is properly partitioned, different parts of the model should be executed in parallel and synchronised. This section explains the need of synchronisation among parts of the model and motivates the chosen synchronisation method.

Need of Synchronisation If we think in network terms, synchronisation among the different parts of a network is needed because:

1. The communication medium is common. This means that the medium access among neighbouring parts of the network has to be synchronised to guarantee correct simulation results.

2. There is inter-partition communication. The network was initially a whole and it is quite possible that a source and a destination node of an existing communication flow are now in different partitions. So, packets among different network parts may be exchanged.

Actually the first argument is a superset of the second. In other words, if the access to the medium among neighbouring parts is properly synchronised, the exchange of packets among them will be also correct, because it occurs at a higher network layer. So, the synchronisation of the medium access is necessary for correct simulation. If two pairs of nodes, placed within a transmission range, try to communicate simultaneously a collision should occur. If these two pairs were assigned to different LPs and the medium access was not synchronised the collision would not occur and the simulation would be wrong. Note that synchronisation is needed only if the partitions are close to each other and can communicate. If two partitions are far away and can not sense and disturb each other they can be simulated completely independent.

The need of synchronisation can be more precisely explained by the use of simulation terms, especially the affect property. It is a feature of events but is also valid for nodes in the network and for LPs. A node affects another node when an event in the first node affects an event in the second one. An LP affects another LP, when a node in the first one affects a node in the second one. But a node in ns-2 affects another node only when it sends a packet, since the medium is the only common resource to the nodes (as discussed in section 2.4). So, an LP affects another LP only when a node on the border sends a packet i.e. when it sends a packet in the common part of the medium.

So, synchronisation of medium access in neighbouring parts of subnetworks is a necessary and sufficient condition for a correct distributed simulation. Therefore distributed simulation of wireless networks needs a synchronisation algorithm to ensure a proper execution of simulation models.

Choice of Synchronisation Algorithm There are two general approaches to synchronisation in event-level distributed simulation — Conservative and Optimistic. Table 4.1 shows the criteria which motivate the choice in this thesis. The "V" sign means that a method satisfies a criterion, and the "X" sign means it does not satisfy it. Next, I discuss the criteria, listed in this table and explain why the conservative method is more appropriate in this thesis.

Feature \ Method	Conservative	Optimistic
Expected speedup	V	X
Straightforward implementation	V	X
Available support	V	X

Table 4.1: Conservative vs. optimistic synchronisation

- Expected Speedup

  I estimate that optimistic synchronisation would have a higher speedup for distributed simulation of *general purpose wireless networks*. The first reason is that lookaheads, extracted from detailed models of wireless networks are small. Efforts to extract lookahead from these models show that in a simulation with 1000 nodes around 90% of the widths of safe windows are in the range of $[5...60\mu s]$ [24]. This simulation time is smaller than the real time for communication between two LPs in a Fast Ethernet network. This means that LPs would spent more real time to determine a safe window, than the length of of the safe window itself in simulation time. Therefore if lookaheads of this size are used with the standard Null-message conservative algorithm[12], the advance in simulation time might be slower that real time. This is not a very promising expectation, because some sequential simulations in ns-2 already run several magnitudes slower than real time (see section 1.2). Another reason for a higher speedup of an optimistic method is that it may make use of a bigger part of parallelism in the model. For example it is very likely that different parts of the simulated network communicate relatively rare, compared to communications within these subnetworks. Then an optimistic method might make use of this parallelism and simulate independent parts of the network in parallel. Note that a conservative synchronisation algorithm can not make use of this kind of parallelism. If internetwork communications are much lower that intranetwork, then a speedup is theoretically possible. But the extraction of this knowledge for a conservative synchronisation requires a deep analysis on dependencies between events which is hardly possible in a complicated model like ns-2.

  However, the models that are used in the context of this thesis are not of general purpose wireless networks. They use a protocol for real-time communication which uses deterministic time slots to access the medium (see section 2.5 for details). This means that every node sends packets and affects other nodes in fixed time intervals. These intervals are in the range of $[1...30ms]$, which allows to extract much larger lookaheads than in the case of general wireless networks. Another important feature of this protocol is that it is *pro-active*. This means that it uses the communication medium even if there is no application data to transmit. In these free time slots the wireless nodes exchange information, used

to maintain a global knowledge for the topology of the network.

These properties of the model add some arguments, which give a better promise from a conservative synchronisation method. First, the lookaheads in the model are much higher than in the general case. This will result in a higher speedup from a conservative method. The second reason is that the pro-active nature of the communication protocol brakes the assumptions of an optimistic synchronisation method. The applications in different subnetworks may still communicate rarely than applications in the same subnetwork. But the underlying pro-active communication protocol still accesses the medium periodically and independently from the needs of the applications. This guarantees a need of periodical synchronisation and limits the optimism of the model.

For these reasons I expect that a conservative synchronisation method will result in a higher speedup than an optimistic one in the context of this thesis.

- Straightforward Implementation

  This criterion shows which method is technically easier to implement.

  The conservative method is closer to a sequential discrete-event simulation than the optimistic one. This is because it schedules all events for the future and does not break the local causality constraint in the simulators. However it requires the computation and extraction of lookahead from the model, which makes it not transparent to the model.

  On the other hand, the optimistic method is well known in the area of distributed computation and simulation as a very complex method. It requires technically complicated operations for state saving, rollbacks, memory management, maintaining multiple lists of events, messages and anti-messages, buffering of I/O operations.

  The computation and extraction of lookahead is more straightforward than the implementation of all these components of an optimistic simulator. Therefore a conservative method wins also by this criteria.

- Available Support

  Another criterion in favour of conservative synchronisation is an available middleware for conservative simulation — `libSynk`[4]. `LibSynk` provides basic services to execute distributed simulations. These include exchange of messages, calculation of safe windows (given the lookahead of the model), time ordered delivery of messages to the simulation executive. Another service is a publisher/subscriber like interface to modify and receive modifications of shared objects in a parallel/distributed simulation.

`Libsynk` has proved its applicability in other projects, including PDNS. It is used there to achieve a conservative synchronisation. PDNS is also an example use of `libSynk` in ns-2, which is an additional support for this thesis task.

The conservative method satisfies all criteria, seen so far, better than the optimistic one. Therefore I choose a conservative synchronisation method in this thesis.

4.4.3 Method for Message Exchange

The last block of the overall system design, not yet described is the "Message Exchange". The task of this component is to deliver messages among the LPs. Remember that logical processes correspond to physical processes in the real network, and a physical process in the context of this thesis is a subnetwork. So, an LP sends a message to another LP, when a subnetwork sends a packet to another subnetwork. Using simulation terms, an LP sends a message to another LP when an event in the first LP affects events in the other one. LPs exchange messages also to realise the distributed synchronisation algorithm. These messages contain upper bounds of safe windows, and other synchronisation information.

The component "Message Exchange" has to define a *notification method* and a *communication protocol*, used in the distributed simulation system. The notification method defines how a receiver LP handles incoming messages. The communication protocol is a transport protocol, that is used to exchange messages among the LPs. Next, I discuss the choice of a notification method and a communication protocol in this thesis.

Notification Method Generally speaking, there are two methods for notification in communication systems: *asynchronous* and *synchronous*. Asynchronous notification occurs when the sender notifies the receiver, while the receiver is doing some unrelated operations. This method is event based and is implemented in computer systems using interrupts. The asynchronous nature comes from the fact that the receiver becomes a message as soon as it arrives, and in a moment when it does not expect it. Synchronous notification is synchronised with the operations of the receiver. Here, the receiver becomes a message only when it requests (polls) to be notified about it. Synchronous notification can be realised in different ways. One of them is to check for new messages (poll) periodically. The alternative is to poll for new messages more frequently, i.e. once per event processing loop, or via busy waiting. Table 4.2 lists these notification methods together with some criteria, that determine my choice. Again the sign "V" means that a method satisfies a criteria, and the sign "X" means that it does not satisfy it. Next, I discuss the particular criteria and motivate the choice for a notification method in this thesis.

Feature \ Method	Asynchronous	Synchronous	
		periodical	frequent polling
Low latency	V	X	V
Implementation and Integration	X	V	V
Available support	X	V	V

Table 4.2: Synchronous vs. asynchronous notification methods

- Low Latency

  This criteria shows whether the notification method delivers messages instantly, or adds an additional delay to the network transmission time. The lower the latency from the notification method, the better, because messages contain information about other simulators. This information may be useful to increase the parallelism and the speedup from a distributed simulation.

  Asynchronous notification has a low latency, because it delivers messages as soon as they arrive. Synchronous notification has also a low latency, but only in its "busy waiting" variant, because of the frequent polling. Periodical polling might delay the delivery of messages until the next poll, and therefore has a higher latency.

- Implementation and Integration

  Synchronous notification methods are easier to implement than asynchronous ones. This is because asynchronous methods introduce racing conditions in memory access. Access to common memory objects during the operation of a process and during a message handler need to be synchronised.

  The network simulator ns-2 is a sequential synchronous system. Therefore it does not provide methods to avoid racing conditions in concurrent data access. So, it is easier to integrate a synchronous notification method than an asynchronous one.

- Available Support

  `LibSynk`, a library for communication and synchronisation in distributed applications provides a synchronous API for message exchange. It is convenient to be used in this thesis for communication and synchronisation purposes. This is another reason to choose a synchronous notification method in this thesis.

The synchronous notification method with frequent polling satisfies all these criteria. Therefore I choose to use it in this thesis.

Feature \ Protocol	TCP	UDP	XTP	RUDP
Reliability	V	X	V	V
Expected performance	X	V	VV	V
Availability (API)	VV	VV	X	X
High level abstraction	VV	X	X	X

Table 4.3: Choice of communication protocol

Communication protocol The final step in the design of the "Message Exchange" component is to choose a communication protocol. This protocol should provide a guaranteed ordered delivery of messages among LPs. This is a requirement of the conservative synchronisation method. Other desirable features are high bandwidth utilisation and low latency communication in a Fast Ethernet network. Such properties would speedup the interactions among logical processes and reduce the running time of a distributed simulation. Table 4.3 lists several reliable communication protocols together with criteria for the choice in this work. The meaning of the sings "V" and "X" is the same as in previous discussions. The sign "VV" means that a protocol satisfies a criteria very well.

TCP and UDP are well known transport protocols, widely used in the Internet. XTP (Xpress Transport Protocol) is a reliable high speed transport protocol, based on [33]. It can operate over the network layer, or directly over the data link layer for lower latency. RUDP (Reliable UDP) is a reliable transport protocol, based on the RFC 980 and RFC 1151. It runs over UDP/IP, but supports reliable ordered delivery, flow control and congestion control mechanisms, similar to TCP.

- Reliability

  Reliability here means a guaranteed and ordered delivery of network packets. This is the most important requirement to the communication protocol, because the synchronisation algorithm relies on it to keep the integrity of the model. All the protocols TCP, XTP and RUDP are reliable by design. The UDP protocol is unreliable, i.e. packets may be dropped or delivered out of order. However a switched Ethernet network provides only one path for packet delivery, so packets can not be reordered. Ethernet has also almost zero error rate, so packets can be corrupted only with a very low probability. But in order to guarantee a 100% reliability, the application has to implement error detection and flow control mechanisms on its own. The design and implementation of these mechanisms in one communication protocol are a relatively complex task. Moreover they deviate from the topic of this thesis. Therefore I prefer to use a ready solution for a communication protocol, and to avoid a proprietary implementation of these mechanisms.

- Expected Performance

  Performance here means high bandwidth utilisation and low latency in a Fast Ethernet network. The XTP protocol wins in this criteria, because it is designed for high-performance communications. It can operate directly over the data link layer for a lower latency. I expect that RUDP and UDP have a lower latency than TCP, because they have a simpler operation. The TCP, even though it is designed for much slower networks, has a high bandwidth utilisation over error-free high throughput links, used nowadays[32]. A disadvantage of TCP is that it introduces latency to the communication. It uses a *congestion control* mechanism, that aims fairness with other communication flows. Therefore TCP delays the sending of a packet, if a congestion may occur. Another source of latency is the *flow control* mechanism. Its goal is to send data only when a receiver is able to process it, and not to overwhelm it. Therefore TCP implies a higher latency than the other protocols. But if a higher latency is a cost for a fair network access among LPs in a distributed simulation, it may finally result in a good overall performance. This is because the simulators would evolve with a similar speed and keep a small time difference.

- Availability

  Availability is the possibility to use a communication protocol. TCP and UDP are widely used and available in all modern operating systems. The XTP protocol has a commercial implementation and therefore is not available for this thesis. RUDP is defined in an RFC and does not have a widely-available implementation.

- High Level Abstraction

  `LibSynk` [4] is a higher level interface to TCP than the standard socket interface. It provides a publisher/subscriber interface to create distributed objects, to update their state and to receive state updates from other simulators. The underlying building of TCP connections, setting of TCP performance options and maintenance of socket descriptors is hidden from the application programmer. Moreover `libSynk` provides the same programming interface for communication both via shared memory, and via TCP. This allows to run the same simulation executive in parallel or distributed simulation, by changing only an environment variable.

TCP satisfies most of the above criteria. Its latency might result in a suboptimal performance, but its availability and higher level abstraction make it attractive to begin with. For these reasons I choose the communication protocol TCP for message exchange in this thesis.

4.5 Design Idea: Replicated Simulation

This section describes an idea for a design of a distributed simulation. It first motivates the need for the idea, then explains the idea itself, and finally analyses its contributions to the thesis.

Motivation Management of distributed simulation has a big influence to its speedup. It includes time synchronisation and exchange of messages among the simulators. If the time spent in management work is too big, the efficiency of distributed simulation can decrease, and the speedup can be even negative. This motivates me to find an idea, or method to decrease management work during distributed simulation.

Idea Lets look at the partitioned graph of one wireless network (see figure 4.3). The graph represents wireless nodes (the vertices), and dependencies between them (the edges). Additionally, the closed regions around each partition show the total joint sensing range of all nodes in the partition. This range determines all other network nodes, that can affect nodes in the partition. For instance, node C lies in the joint sensing range of partition 1, and can affect the nodes A and B. In the same way nodes A and B lie in the sensing range of partition 2, and can affect node C. For this reason, simulator 1 has to be aware of node C, and inform simulator 2 each time when node A or B send a packet in the network. Also, when node C sends a packet in the network, simulator 1 should have some information about node C in order to compute the effects from this transmission on the nodes A and B. This requires that simulator 1 has some representation of node C, which allows it to compute its influence on the local simulation. This representation has to be regularly updated with the true copy of node C at simulator 2, in order to achieve a correct simulation.

Now, let simulator 1 also maintain a full instance of node C, which is a copy of node C in simulator 2, and behaves in the same way in both simulators. Then, when node C sends a packet, simulator 2 no longer needs to inform simulator 1 about it, because simulator 1 has also evolved to this transmission via a local computation. In the same way, simulator 2 can maintain instances of the nodes A and B. So, when they send packets and affect nodes in partition 2, simulator 2 "knows" about this transmission and need not to be informed.

So, the idea is to *replicate* the simulation of *border nodes* in all simulators, which they can affect. The replication has to maintain an instance of a border node in all these simulators. This instance has to be the *same*, and have the *same behaviour* in all simulators in order to achieve a correct simulation. In this way the activities that border nodes initiate (e.g. send a packet) do not need to be synchronised among different simulators. This could reduce the synchronisation overhead in a distributed simulation. Figure 4.4 represents this idea graphically.

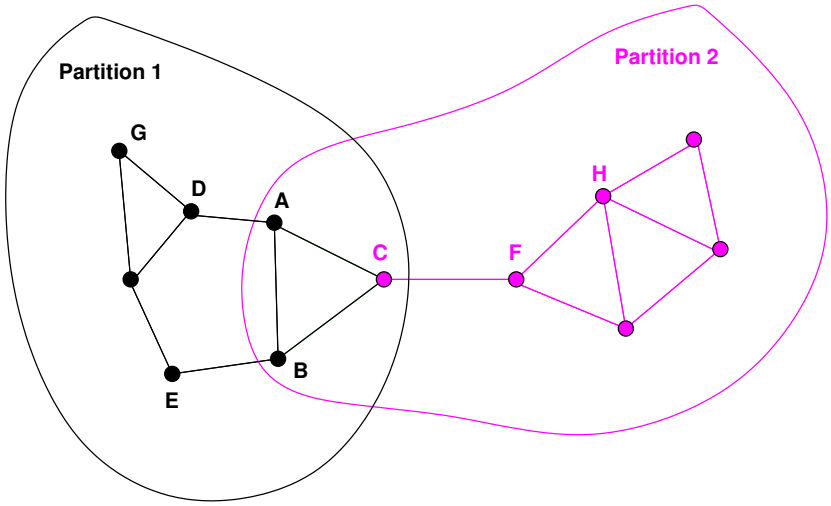


Figure 4.3: Partitioned graph with sensing ranges

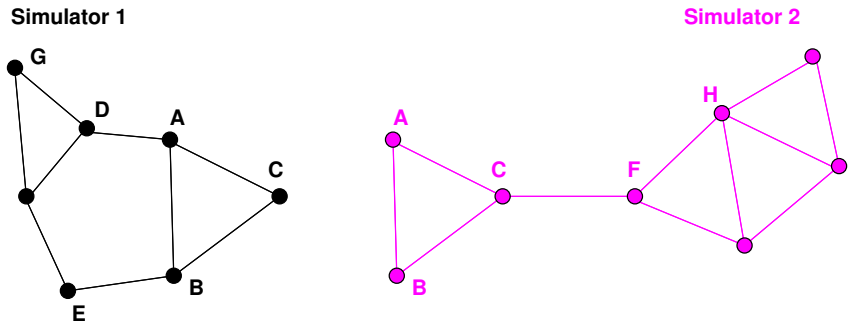


Figure 4.4: Replicated simulation of border nodes

Since every node in the overlapping area is presented in both simulators, there is no need to synchronise the activities of the nodes A, B, and C. However, packets that come from a non-overlapping area to an overlapping area have to be synchronised. These are packets sent in the directions $D \rightarrow A$, $E \rightarrow B$, and $F \rightarrow C$. If these packets are properly synchronised, the behaviour of the border nodes A, B, and C will be the same in both simulators. In order to prove this I consider the group of nodes $\{A, B, C\}$ as a finite automaton. The inputs to this automaton are packets, that come from outside, i.e from a non-overlapping area. The states of this automaton are all possible combinations of state variables, that represent these three nodes in the model. The outputs of the automaton are packets, that come out of it over the logical links $A \rightarrow D$, $B \rightarrow E$, and $C \rightarrow F$. If the initial states of this automaton are the same in both simulators 1, and 2, and these two automatons receive the *same* input signals, then their states, and outputs will be the same during the whole simulation. Therefore, it is enough to synchronise packets on the input logical links (i.e. $D \rightarrow A$, $E \rightarrow B$, and $F \rightarrow C$) in order to achieve the same states and behaviour of border nodes.

Analysis In a late stage of the thesis project I realised that this idea actually does not necessary reduce synchronisation overhead in wireless networks. The main aspect, that is neglected by the idea is that wireless networks have a *broadcast* nature. This means that even if node F sends a packet to node H, node C will also be affected. When the method for replication of border nodes is used, this will result in an additional synchronisation, which is not be needed in the case without replications. So, this idea on the one hand decreases synchronisations in the border area, but on the other hand increases synchronisations in the areas near the border. This means that it changes the boundaries, where synchronisations occur, but no necessary decrease them.

This idea does not show promising results in the distributed simulations of wireless networks. But it is possible that it can be applied to other kinds of distributed simulations — wired networks for example. The mathematical proof of the idea for replicated border simulations and a test for its applicability in other models can be an interesting topic for further investigations.

Nevertheless, this idea has also a positive contribution to the thesis. Now the interactions, that happen in the model, and need to be synchronised can be completely computed by a simulator locally. This means that when a node sends a packet, and this transmission has to be synchronised with another simulator, all destination nodes are also available in the source simulator. Therefore the simulator can compute all necessary parameters for a transmission locally using the ready and implemented methods of the sequential simulation. These parameters are propagation delay and reception power of the signal at the receiver side. The use of the implemented methods in the sequential simulation has two advantages. Firstly, it is secure, i.e. it leaves model-related computations to the sequential model, and avoids possible errors. And

secondly it makes the design of the distributed simulation simpler. Now, the distributed simulation does not go into deep model-related details of the sequential model. The disadvantage of replicated simulation is clear: it introduces redundant simulations and computations for the border nodes. But since the border nodes in large network models are a small part, this effect can be tolerated.

4.6 Design Interpretation

This final section of the design summarises its main contributions to the thesis in form of advantages and disadvantages. Finally it estimates the feasibility of the implementation, which is the following step in the thesis, based on the design.

Advantages

- Modular Overall Design

  The designed parallel/distributed simulation system has a block design, and the interfaces among the blocks are clearly defined. This allows to change/upgrade a component of the system without the modification of the other ones. For instance the component for message exchange can be changed transparently to other components of the system. If the used communication protocol results in a low performance, it is enough to develop a module for `libSynk`, to use another lower latency protocol.

- High Quality Partitions

  The component for model partitioning uses graph partitioning techniques to divide the initial model into equal parts. This method promises high quality partitions at a reasonable cost. This is because the graph partitioning problem is well known in the graph theory and implementations of such algorithms exist[26, 25, 27, 5].

- Optimal Synchronisation Method

  The design suggests a conservative synchronisation method in the distributed simulation. It promises a better speedup in the context of this thesis, because of timing properties of the modelled communication protocols. A conservative synchronisation method is also easier to implement than an optimistic one, moreover PDNS is an existing example for it.

- Abstraction from Communication Method

  The decision for a communication method is to use the `libSynk` middleware[4]. This is a communication and synchronisation layer for distributed applications. `LibSynk` provides transparency from the underlying communication protocol, and

the resulting simulation executive can use both shared memory and a network for communication. This allows to use the same simulator for both parallel and distributed simulation.

- Interesting Design Idea

  The design of this thesis introduced an idea for replicated computations in distributed simulation in order to reduce synchronisation. Even though this idea does not promise reduce of synchronisation in this thesis, it might be an interesting topic for research in distributed simulation. Moreover, other research work has shown that replicated computations can reduce synchronisation in distributed systems[30].

Disadvantage

- Non-optimal communication performance

  The design suggests TCP as a communication protocol, because of its availability and ease to use. However, TCP introduces latencies that may result in a suboptimal performance.

Feasibility of Implementation The design suggests reuse of other efforts in this thesis. These are mainly graph partitioning algorithms and a communication/synchronisation middleware. Graph partitioning algorithms have a high complexity, because they use complex mathematical and combinatorial operations. Their implementation would be a task for another master's thesis. The synchronisation algorithms are also complex, because of their distributed nature. In spite of the complexity of these algorithms, the reuse of other efforts in this thesis keeps their realisation feasible.

However, the implementation still has its challenges, which have more or less a technical nature. One of them is to keep the integrity of a model by a distributed execution of ns-2. This means to replace interactions between subnetworks in a sequential simulation by interactions between simulators in a distributed simulation. Another challenge is a method to use a dynamic lookahead in the simulation model, for an optimal performance of the synchronisation algorithm. These questions will be answered in the next section.

5 Implementation of the Simulation System

Following the design, described in the previous section 4, this section describes the realization of the distributed simulation system. It starts with a description of the implementation process in 5.1, and an overall view of the implementation in 5.2. Then, section 5.3 describes in detail the operation of different blocks of the system. It is mainly focused on the partitioning and distributed execution of the simulation model. Finally section 5.4 summarises the achievements of the implementation and estimates its contributions to the work.

5.1 Implementation Strategy

Section 4.3 already introduced the overall system design, and the basic blocks and interfaces in the system. This section follows this design and suggests a strategy for its implementation. The strategy is graphically represented on figure 5.1. It is to divide the system into smaller subsystems, where each subsystem has one input interface and one output interface. Each subsystem is then implemented separately and at the end they are merged to construct the whole system.

The overall design suggests three basic blocks and four interfaces (see figure 5.1). This implies three subsystems, each having one input and one output interface. This implementation strategy gives a flexibility to change modules of the system, but also a requirement to keep the interfaces between different blocks clearly defined.

The implementation strategy divides the work into three stages. The first is to implement the partitioning of the input model, and to produce an input for the next stage. This block aims to divide the model into equal parts and minimise the amount of communications in a distributed execution. The second stage is to realize a distributed execution of the simulation model. It should use a synchronisation algorithm to keep the integrity of the model, and produce the same results as a sequential simulation. The distributed execution provides a simulation result for each sub-model. The final stage of the work is to implement the module for merging partial results into a final simulation result, corresponding to the input model. The next sections give an overview of the implementation, and then a detailed description of each stage.

5.2 Implementation Overview

Figure 5.2 shows an overview of the implementation of the distributed simulation system. The system has three parts, as suggested by the design: preprocessing (model partitioning), processing (distributed simulation), and postprocessing (merge of results). The main component "Distributed Simulation" is further divided in 2 parts: "Distributed Execution" and "Synchronisation and Message Exchange". The blocks for synchronisation and exchange of messages were suggested as separate ones

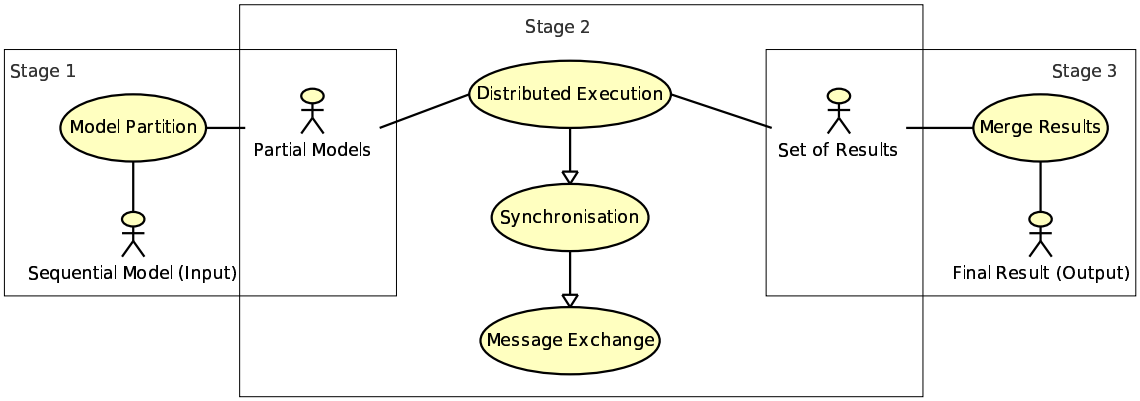


Figure 5.1: Implementation strategy

by the design. However, the implementation merges them into a single one, because the `libSynk` library provides a common API for both operations. `LibSynk` is a library for synchronisation and message exchange in parallel/distributed application, which is also used in this thesis. It provides timely ordered message exchange and time management for synchronisation in distributed simulations[4].

Figure 5.2 shows the main components of the system, and the interactions between them. The class `Simulator` is a part of ns-2, and it defines and configures the simulation model. In this implementation it additionally initiates a partitioning of the model, before its execution, and also a postprocessing of the results after the execution. To partition the model, the `Simulator` uses the class `GNetwork`. Its task is to acquire information about the model from ns-2, represent it as a graph and run a graph partitioning algorithm using the `Metis` library[5]. `GNetwork` also activates all nodes that are assigned to the current simulator, and deactivates other nodes in the model. So, when the model is partitioned and only one part of it is active it is time to execute it. Then the `Simulator` gives the control to the `Scheduler` until the end of the simulation. The `Scheduler` controls the distributed execution of the model, by maintaining and running local events, and receiving messages from other simulators. The components `Ns-2 Object` represent structures that build the model in ns-2, e.g. network nodes, network layers, timers. These objects receive the control from the scheduler, execute a step in the simulation model, generate a history about it, and possibly send messages to other simulators. They use the component `CommunicationInterface` to transmit these messages. The `Scheduler` also uses this component in order to receive messages from other simulators and to advance simulation time. The `CommunicationInterface` is an interface to the `libSynk` library. At the end of the simulation the `Simulator` uses the block for merging of results to construct a single simulation result from multiple

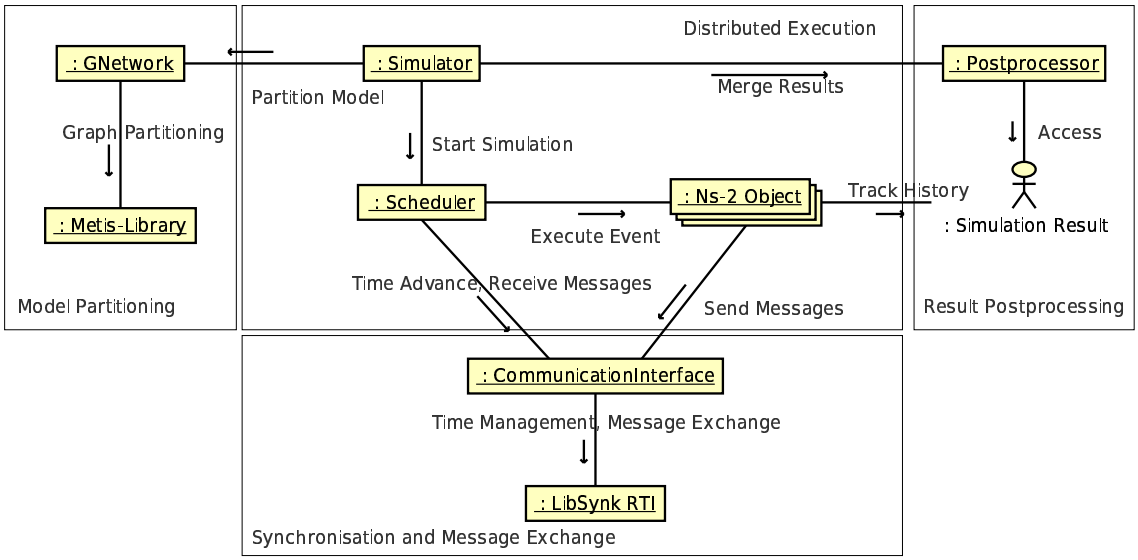


Figure 5.2: Implementation overview

partial ones.

5.3 Implementation of System Components

This section describes in detail the implementation of the different building blocks of the system. It starts with the partitioning of the simulation model in 5.3.1. Then section 5.3.2 describes the distributed simulation of the model. This is the main part of this work, and also the main part of this section. Finally, section 5.3.3 describes postprocessing of simulation results from the different sub-models, which completes the description of the system.

5.3.1 Model Partition

The subsystem for model partitioning divides a sequential network model into multiple sub-models, suitable for distributed simulation. As discussed in the previous section 4.4.1, the design of the system suggests to partition the model by the network nodes. So, the input interface of this block is a description of a network model in the simulator ns-2. The output is an assignment of each node in the network model to a *network partition*. Network partition is a group of network nodes, that is considered as a logical process (LP) in the distributed simulation, and is simulated by a separate simulator on a separate processing unit. The used method for model partitioning, suggested by

the design, is to partition the input network with a graph partitioning algorithm. This method requires two steps:

1. Represent a wireless network in the model of ns-2 as a graph
2. Partition the graph using a graph partitioning algorithm

Graph Representation This step reads the network topology of ns-2 and creates a graph $G(V, E)$ from it. $V = \{v_i | i = 1...|N|\}$ is a set of vertices in the graph, which correspond to nodes $N = \{n_i | i = 1...|N|\}$ in the network model. $E = \{e_i | i = 1...|E|\}$ are edges in the graph, that represent a possibility that one node *affects* another one in a simulation. Two vertices v_k and v_l are connected by an edge e_{kl} if the corresponding nodes n_k and n_l can affect each other in a simulation. Network nodes n_k and n_l can affect each other in a simulation if they:

1. Use the same wireless channel
2. Lie within a sensing range (see section 2.4 for more details).

This means that if n_k can affect n_l, then n_l can also affect n_k, and therefore the graph $G(V, E)$ is undirected.

The network simulator ns-2 maintains an internal list of all network nodes. This list has a memory management function and does not show the connections between nodes in a wireless network. The set of vertices V is read from this list. The set of edges E is determined by inspecting each network node and defining its neighbour nodes, using the rules above.

Graph Partitioning The graph partitioning problem is well-known in the graph theory[25]. It is formally defined as follows. Given a graph $G(V, E)$, with set of vertices V, and with set of edges E, partition V into k subsets $V_1, V_2, ..., V_k$ such that:

1. $V_i \cap V_j = \oslash \mid \forall i \neq j$
2. $\bigcup_{i=1}^{k} V_i = V$
3. $|V_i| = \frac{|V|}{k}$
4. Number of edges, connecting vertices in different subsets is minimal

This means to divide a graph with $|V|$ number of vertices into k subgraphs, such that they do not overlap, and their union forms the initial graph. The other two conditions require that the number of vertices in the different subgraphs are equal, and the number of edges between subgraphs is minimal.

There are lots of different methods for graph partitioning, known in the graph theory: geometrical, combinatorial, spectral, multilevel. The study [27] makes an overview of different techniques and compares them quantitatively. It suggests that a combination of multilevel and combinatorial method, called Multilevel k-way is suitable for graph partitioning in scientific parallel/distributed simulations. It produces high quality partitions, at a lower cost, compared to other partitioning methods.

The Multilevel k-way partitioning algorithm operates in three phases. First it coarsens the input graph, i.e. it reduces its size by combining vertices together, and accumulating information about vertices and edges. Then it partitions the coarsened graph using a multilevel bisection algorithm[25]. The final phase of the algorithm is a refinement process, which maps the partitioning of the small graph to the initial graph. In this final phase the algorithm uses combinatorial optimisation criteria to maximise the quality of the partitioning. The Multilevel k-way partitioning algorithm is described in detail in [26].

The Metis library [5] implements a Multilevel k-way partitioning algorithm, and I am using it in this thesis. It uses an optimisation criterion for the partitioning, especially suited for parallel computing. The criterion is to minimise the total communication volume *totalv* between processors. It can be described as follows. Let the initially partitioned graph be $G(E, V)$, and let the set $V_b \subset V$ contains all border vertices, i.e. all vertices that have an edge to another partition. For each border vertex $v \in V_b$, the array $Nadj[v]$ contains the *number* of different partitions, which v is connected to. Each border vertex $v \in V_b$ has also a weight w_v, which describes the amount of communications, needed to synchronise the sate of the vertex v with another processor. Then the total communication volume is defined as:

$$totalv = \sum_{v \in V_b} w_v * Nadj[v]$$

The Multilevel k-way partitioning algorithm tries to minimise *totalv* by using a combinatorial scheme for moving vertices from one partition to another. This measure of a partitioning reflects the amount of communications, used in a distributed simulation because it counts the number of connections to external partitions. This is different from counting the number of edges to external vertices, because a border vertex may have multiple external edges, but still less connections to external partitions. Connections to external partitions contribute to the amount of communications, because they represent information flow between different processors. This idea is illustrated on figure 5.3. Vertex number 5 has two external edges, but they are to the same external partition. So, even though the graph contains two *model-related* paths from partition 2 to partition 1, information about the state of vertex 5 needs to be transmitted over a single *communication* path between these partitions. Therefore this partitioning objective reflects the communication pattern in a distributed simulation.

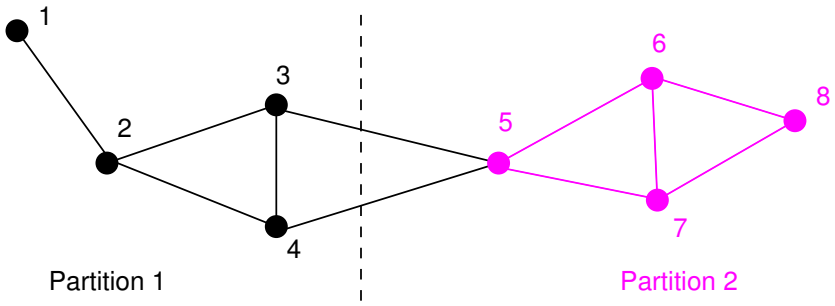


Figure 5.3: Graph partitioning objective

The weights of the vertices have to be determined from the simulation model. The weight w_v of each vertex v is determined by the weight of the corresponding node: $w_v = w_n$. In this thesis the distributed execution is done with copies of border nodes, and synchronisation occurs when a border node receives a packet. So, the amount of communications, which a node introduces in a distributed simulation depends on the amount of data it receives, i.e. the data that its neighbours send in the network. And since the wireless medium has a broadcast nature, all data has to be taken into account, and not only packets, destined to this node. For example, in figure 5.3, if node 6 sends a packet to 7 or 8, then node 5 is also affected. Therefore all the data that node 6 sends in the network has to be taken into account, by computing the weight of node 5. Now, let the amount of data that a node i sends in the network be D_i, and the neighbours of a node n be the nodes in V_n. Then the weight w_n of node n is the sum of data amounts that its neighbours send in the network:

$$w_n = \sum_{i \in V_n} D_i$$

The simulation engine can hardly determine the amount of data D_i for each node before the simulation, and therefore it estimates it. I consider the network as a group of logical links between network nodes and assume that all logical links are utilised with an equal amount of data. Therefore I use

$$D_i = 1$$

for each data link. For example, the weight of node 5 on figure 5.3, based on amounts of data D_i, is given by the expression:

$$w_5 = D_3 + D_4 + D_6 + D_7 = 4$$

Example graph partitions, obtained by the Multilevel k-way partitioning algorithm and these weight settings are shown on figures 5.4 and 5.5. The first figure shows a

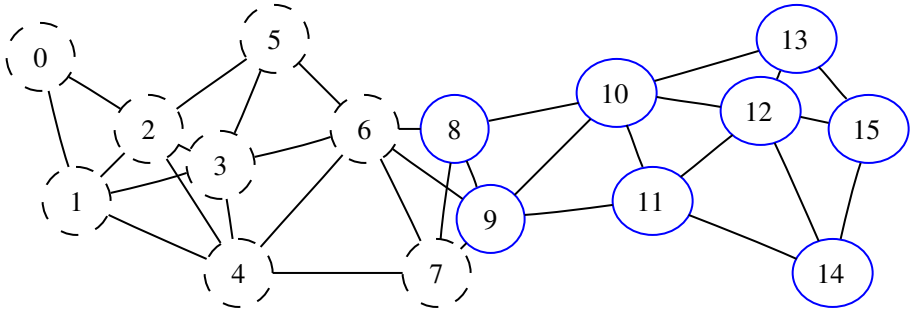
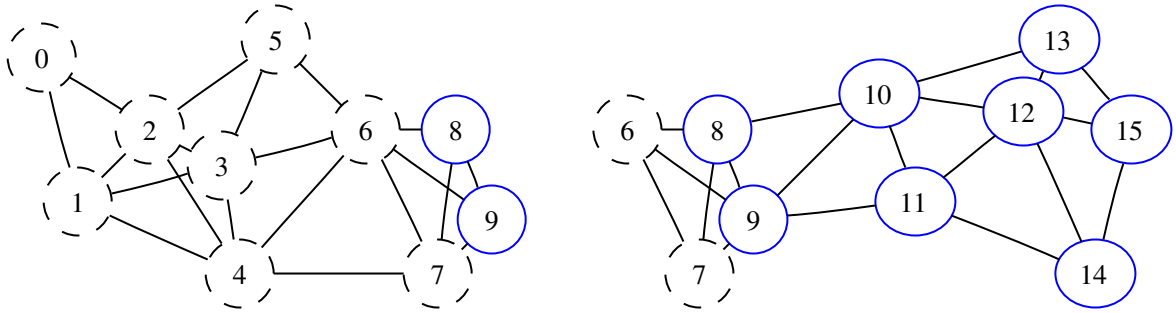


Figure 5.4: Partitioned graph



a) Simulator 1 b) Simulator 2

Figure 5.5: Graph parts for two simulators

partition of a graph, and the second one shows parts of the graph, that are dedicated to two different simulators. Note that the method for copies of border nodes is used here, and therefore each simulator simulates also the border nodes of the other one.

5.3.2 Distributed Execution

The realization of a distributed execution is the second stage of the implementation, and here I explain it in detail. First I follow the design and implement the idea for multiple instances of border nodes. Then I extend the overview of the module for distributed simulation, given in section 5.2. I describe the algorithm of the scheduler for a conservative synchronisation, and explain basic operations in distributed simulation — sending and receiving of messages and advancing simulation time. Finally I give some remarks on lookahead extraction and repeatability of distributed simulations.

Instances of Border Nodes Each logical process runs all nodes in its own partition, and also all neighbouring nodes from other partitions. This means that the nodes on the border have multiple instances in different simulators. The requirement for a correct simulation is to achieve the *same behaviour* of all instances of the same border node

in all simulators. In order to synchronise the behaviour of all instances of a network node, I consider it as a finite automaton. The input signals of this automaton are packets, that the node receives from the network. The outputs of the automaton are packets that the node sends in the network. Since the model of ns-2 is deterministic and repeatable one, the automatons (instances of nodes) can have an equal initial state. So, if their input signals and their stochastic behaviour are synchronised, they will have the same behaviour and produce the same outputs.

So, when a border node receives a packet in one logical process, the simulator sends messages to all other LPs, that maintain an instance of the same border node. Note that this synchronisation is not always necessary, when the sender is also a border node. But there are special cases, which require this redundancy. These will be shown in the experimental part of the work (section 6.3.1).

Since the model of ns-2 contains random numbers, the stochastic behaviour of multiple instances of the same network node should be also synchronised. For example network nodes in ns-2 use random variables to model a probability for transmission errors. Then, if one instance of a node does not receive a packet because of a transmission error, other instances should also not receive it in order to achieve the same behaviour. The ns-2 implementation uses a single random number generator (RNG) for all random variables in the entire model. In order to synchronise the stochastic behaviour of multiple instances of border nodes I introduce a separate RNG for each node. Then I initialise the RNGs of multiple instances of the same node with the same seed, so that they produce equal random number streams.

Another issue that has to be taken into account is the generation of a history of the network from the model. Since multiple instances of the same node execute the same events, there is redundant information at the end. To avoid redundancy I turn off the generation of history on the copies of border nodes. Now only one instance of a border node generates history — this is the instance on the LP which the node is originally assigned to from the partitioning.

Scheduling Algorithm The following discussion focuses on the distributed execution of the simulation model. Section 2.1 illustrated the algorithm of a sequential discrete-event simulator. The algorithm of a distributed DES with a conservative synchronisation has an additional component for handling messages and processing events in time-stamp order. Here, I describe the algorithm of the conservative scheduler, used in this thesis. It is adopted from PDNS[7], which is also a conservative distributed simulator, using `libSynk`[4].

The scheduler, used for a distributed simulation is implemented in the class `FederateScheduler`, which inherits the class `Scheduler` in ns-2. Algorithm 5.1 shows the main working cycle of the `FederateScheduler`. At the beginning it takes the next earliest event `e` and requests to advance its time up to the time of `e`. If

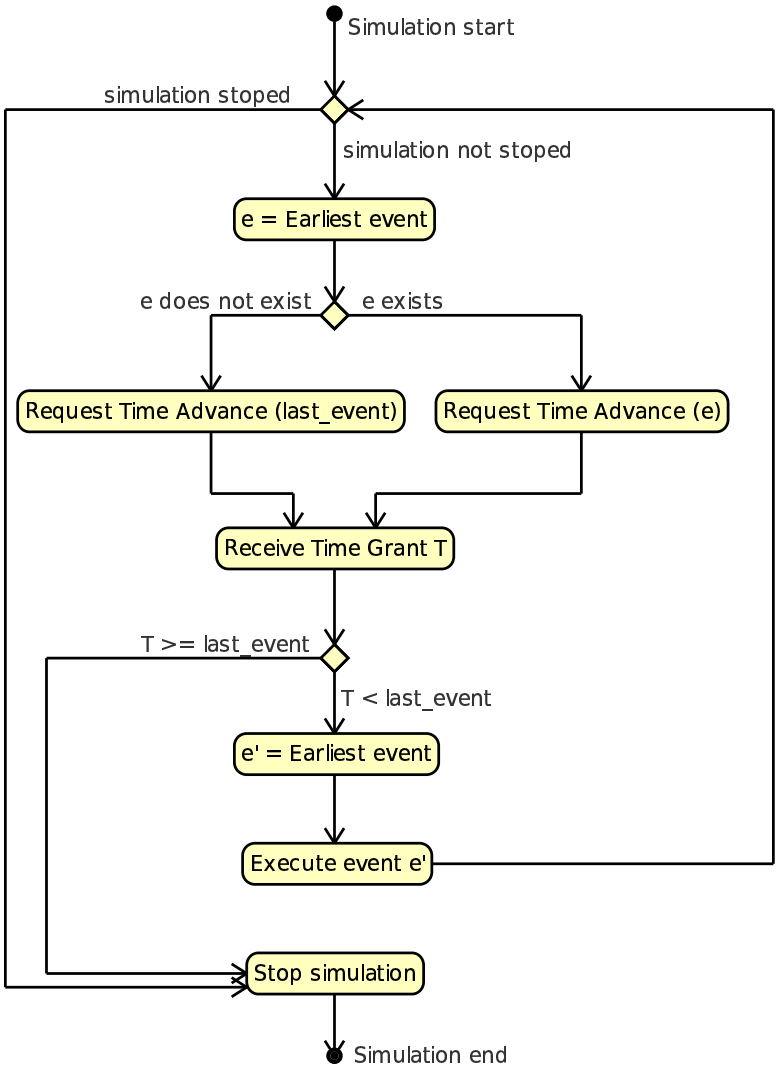
there are no events in the queue, the scheduler requests to advance its time up to `last_event`, which is a preconfigured large time for the end of simulation. The operation `Request Time Advance` is implemented by the `CommunicationInterface` and `libSynk`, and will be described in detail later. After this operation the control returns back to the scheduler, which receives an allowance to increase its time to `T`. If `T` is equal to `last_event`, this means that there are no more events for this simulator in the system and it can stop. Otherwise the simulator continues to execute events. However the granted time `T` is not necessary equal to the time of event `e` and can be smaller. This is because during the operation of time advance it is possible that the communication middleware delivers new messages and inserts events in the queue before `e`. But after giving a time advance grant, the communication middleware guarantees that there is at least one event that can be processed. Therefore, after this operation the scheduler takes again the next earliest event `e'`, and executes it. It is possible that the event `e'` affects parts of the model, assigned to other simulators in the system. In these cases it sends them a message to inform them about the change. For this purpose it uses the operation `Send Message` of the communication middleware. After the control comes back to the scheduler it continues with the next event in the list.

Next, I explain the operations `Send Message`, `Request Time Advance` and `Receive Message` in detail.

Send Message Figure 5.6 shows a diagram of a simulator, sending a message to another simulator. First `FederateScheduler` executes an event, by calling its `Handler`. If this event is at the lowest network layer it may send a network packet to the `WirelessChannel`. This is a class that models the radio channel in ns-2, and basically transfers network packets between nodes. I have extended it to the `WirelessChannelFed`, which takes into account that some nodes are simulated on other hosts and generates messages for them from the ns-2 packets. However, network packets in ns-2 are not represented as sequences of bytes, but are complex structures with pointers. Therefore the `WirelessChannelFed` needs to transform ns-2 packets into sequences of bytes before it that can send them through a real network. For this reason it uses the class `SerialPacket` to transform ns-2 packets into serial messages and reverse.

After the ns-2 packet is transformed into a sequence of bytes the `WirelessChannelFed` passes it to the `CommunicationInterface`. It uses the `libSynk` middleware and updates the attributes of a distributed object. The middleware then decides to which other simulators to send the packet based on a publisher/subscriber information about the distributed object. The distributed object is a part of wireless medium, situated in the overlapping area between two or more simulators. All simulators that access this part of the wireless medium declare to `libSynk` their intention to modify it (publish information). They also state a wish to receive

Algorithm 5.1 Conservative synchronisation algorithm



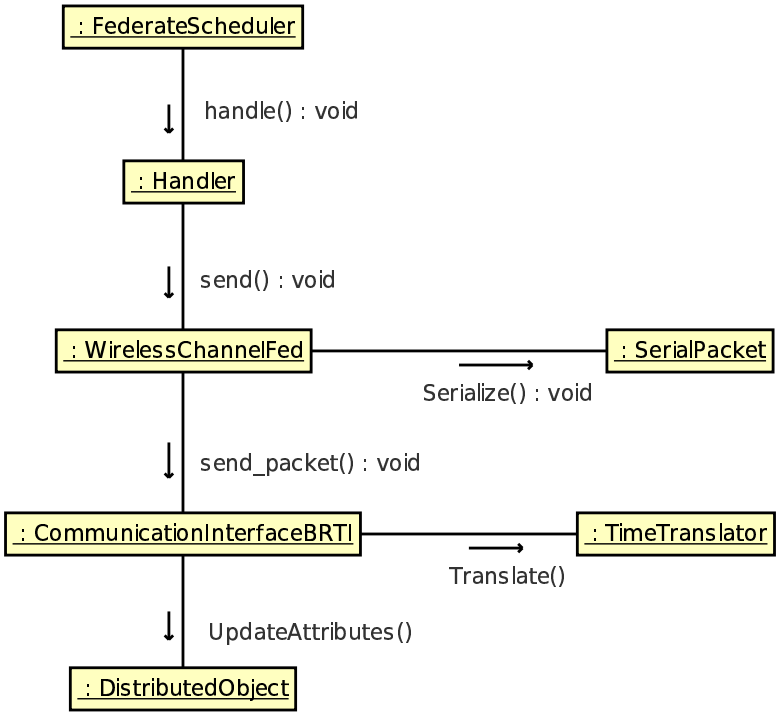


Figure 5.6: Operation `Send Message`

modifications from other simulators (subscribe for it). In this way the `libSynk` middleware has the knowledge to deliver update messages to interested simulators.

Advance Time and Receive Messages These two operations are combined together, because a conservative synchronisation algorithm advances simulation time to a future moment only when it has received all messages until that moment in time. In this implementation the `libSynk` library controls the timely-ordered flow of messages and determines up to what time a simulator can advance its local clock. Figure 5.7 shows a process of time advance and reception of messages.

When the scheduler needs to advance simulation time in order to execute a future local event, it has to make sure that no messages will arrive until that time. So, it gives the control to `CommunicationInterface`, which tries to advance simulation time, using the primitives of `libSynk`. It first declares a request for time advance, and then continuously polls and waits for a time advance grant. If the middleware knows from a previous computation of safe windows that the requested time advance is allowed, it returns immediately and gives an advance grant. But if the requested time advance is not in the safe window, `libSynk` initiates a new computation of

safe windows. After this operation completes, `libSynk` delivers new messages at the next poll via the method `ReflectAttributeValues ()`. This is a callback function of the `CommunicationInterface`. It receives a message, and then calls the `WirelessChannelFed` to generate ns-2 network packets from it and insert them in the local event queue. Finally, after delivering all possible messages until a moment in the future, `libSynk` grants a time advance to the `CommunicationInterface`, which forwards the grant to the scheduler.

The `libSynk` middleware uses a *reduction based* algorithm, and uses *conditional information* to compute safe windows. Initially each simulator sets its safe window to infinity. Then it conditionally guarantees that it will not send any messages until the time:

$$T_{cond} = Time(EarliestEvent) + Lookahead$$

This means that a simulator will not send any messages, with timestamp less than T_{cond}, *under the condition* that it does not receive any new messages until the time $Time(EarliestEvent)$. This is because there are no events before the time of the earliest event. The middleware uses conditional guarantees from all simulators, information about the connections between them, possible message flows, and information about transient messages to compute the safe windows. Transient messages are messages that are sent, but not yet received at their destination. They are an issue by computing safe windows in distributed simulation[18]. The `libSynk` middleware *reduces* the safe windows until it is not possible that a simulator in the system receives a message within its safe window. Then it gives time advance grant until the end of the safe window of each simulator.

Lookahead Extraction Lookahead in a distributed simulation depends on the model, and has to be *extracted* from it, using model specific knowledge. An intuitive source of lookahead in a network simulation is the network propagation delay. This is the time that one bit from the signal "travels" through the medium. Therefore it is guaranteed that an event in one node can not affect an event in another node, before a network propagation time in the future. So, it can be used as a *constant lookahead*. However, propagation delays in wireless networks are in the range $[1...1500ns]$, which is too short to promise large safe windows and a good speedup.

Other properties of the model also exist, that can be used to extract lookahead. These are for example properties on the data link layer in a wireless network. Another sources of lookahead are properties of the real-time communication protocol, used in the context of the thesis. Since it uses a time division access to the medium it is possible to predict an earliest moment in future in which a node may send in the medium. Using these two sources of lookahead, it is possible to determine larger lookaheads in the range $[1...30ms]$. However, these are *dynamic lookaheads*, i.e. they are based on a current state of the model and change as the model evolves in time. Using these

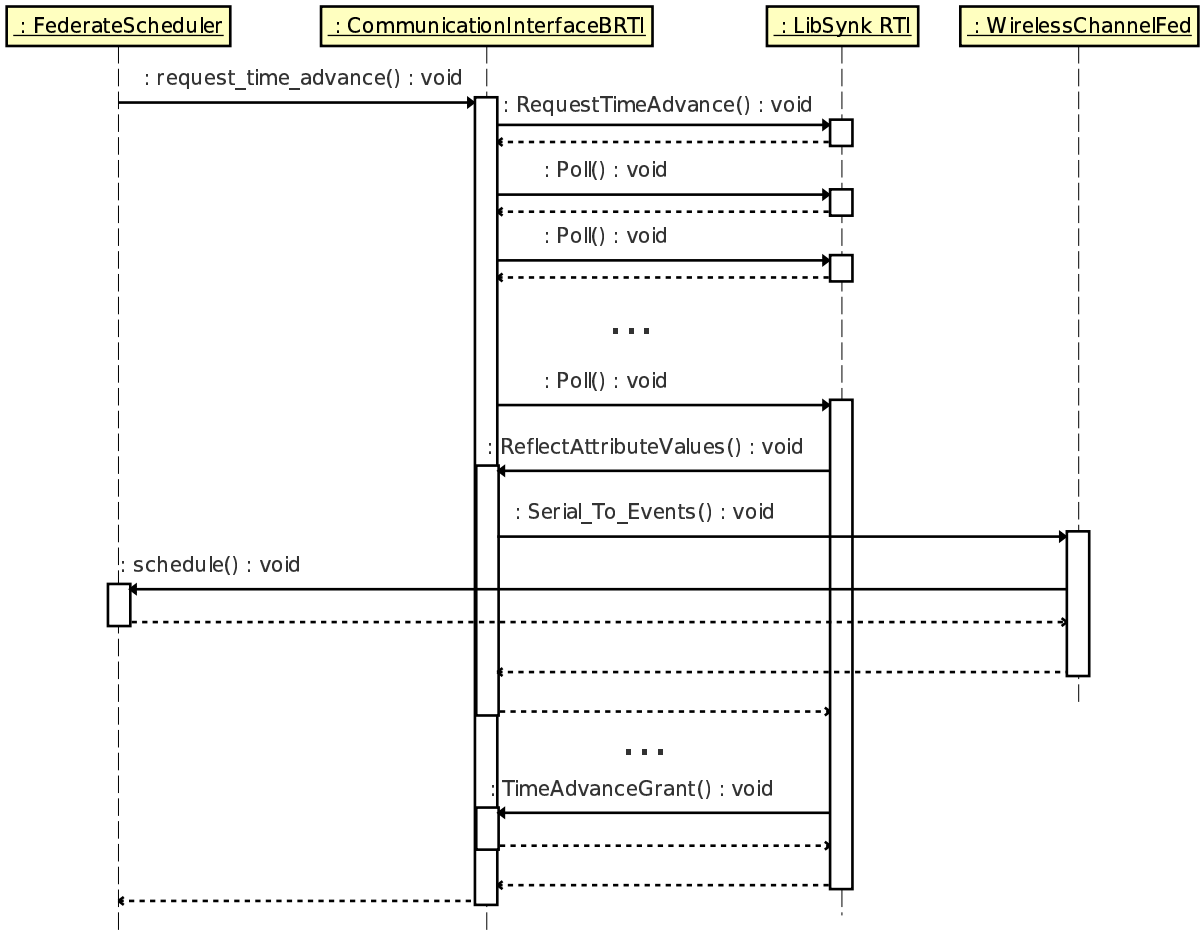


Figure 5.7: Operation `Receive Message and Advance Time`

lookaheads requires a method for dynamically changing lookahead in the distributed simulation.

But the used interface to the `libSynk` library in its current version 1.13 does not provide primitives to dynamically change lookahead during simulation. Therefore I can not use these large lookaheads in the current implementation. An extension to the `libSynk` library is needed in order to use the larger but dynamic lookaheads available in the simulation model. This extension remains a problem of future investigations.

Repeatability Repeatability of distributed simulations is another point, that has to be taken into account by the implementation. Like every other scientific experiment, distributed simulations also have to be repeatable, and produce the same results in multiple executions. A possibility for non-repeatable simulations is introduced by messages with the same timestamp. In order to be repeatable a simulation has to guarantee that it executes local events *in the same order in each run.* The order of execution of events with different timestamps is fixed by the simulation algorithm, but the order of events with *equal timestamp* is not. In sequential simulations, this problem can be easily solved by executing concurrent events in a FIFO manner. However, this is not a solution in a distributed simulation, because messages are transmitted and delayed through a network. Consider that two simulators send two messages with the same timestamp to a third one. In this case a FIFO ordering of the messages by the third simulator is not appropriate. This is because in different executions these two messages may be delayed different amounts of time through the network, and may be received in a different order.

There are known techniques for breaking ties in distributed simulations, e.g. hidden time stamp fields, priority numbers, or receiver specified ordering[18]. In this thesis, I am using the third one, by additionally considering the IDs of the simulators by the order of events in a local queue. It guarantees that distributed simulations with the same number of simulators produce the same results from one run to another. The scheduler now uses a composite timestamp, created from the time of an event, and the ID of a simulator, which has originated this event. If two events have the same timestamp, then the event from the simulator with a smaller ID is processed first. If these two events are received from the same simulator, then they are processed in a FIFO order. The FIFO ordering can not result in ambiguities here, because the communication protocol TCP provides ordered delivery of messages. This guarantees that if two messages come from the same simulator, and have the same timestamp, they will be processed in the order which they were send. And since the model executes in a deterministic way, a simulator will always produce messages in the same order, provided that it processes input messages also in a deterministic way.

The interface to the `libSynk` library, used in this thesis, provides possibility to operate only with timestamps, containing a single field, which is the time from the

model. The additional ordering of simultaneous events by simulator IDs has to be implemented on top of `libSynk`. In order to correctly do this, the `Scheduler` has to know all available events with a next timestamp $T_e = T_{earliest}$ before starting to process the first one. I implement this feature in the `CommunicationInterface` transparently from the `Scheduler`, and from `libSynk` in the following way. When the scheduler requests a time advance up to time T_e, the `CommunicationInterface` requests from `libSynk` a time advance grant of $T_e + \Delta t$ (Δt is a small time). When it receives this grant it means that all messages until time $T_e + \Delta t$ are known to the `Scheduler`, and it has ordered them in its local queue based on their sender ID. Now the `CommunicationInterface` grants the time T_e to the `Scheduler`, and it can safely start to process the first event in the row of events with at time T_e.

Note that when the CommunicationInterface requests a time $T_e + \Delta t$ from `libSynk` it conditionally guarantees that the simulator will not send any messages with a timestamp less than $T_e + \Delta t + Lookahead$. But it is possible that an event at time T_e generates a message with timestamp $T_e + Lookahead$ which will not be accepted by the synchronisation mechanism. Therefore the `CommunicationInterface` decreases the lookahead at $Lookahead - \Delta t$ in order to ensure the correct operation of `libSynk`.

This measure needs a further refinement, when it is applied in practice, because of floating point roundoff errors. This is because the model computes the time of the next event by:

$$T_{next} = T_e + Lookahead$$

and the `libSynk` computes a conditional guarantee for no messages, sent until the time:

$$T_{guarantee} = (T_e + \Delta t) + (Lookahead - \Delta t)$$

It is possible that $T_{next} < T_{guarantee}$ due to floating roundoff errors. Therefore, the CommunicationInterface further decreases the lookahead at $Lookahead - 2 * \Delta t$. The choice for $\Delta t = 1e^{-12}$ is sufficiently big to compensate roundoff errors and guarantee that $T_{next} > T_{guarantee}$. On the other hand it is sufficiently small (1 picosecond), and does not have a significant impact on the performance of distributed simulation.

5.3.3 Postprocessing of Results

After the distributed simulation has finished, each simulator provides history of the sub-model which it has simulated. It is in a form of a text file, where each line contains information about an event, e.g. "a node has sent a packet at time T_1", or "a node received a packet at time T_2". The events, written in each are sorted by their timestamps.

Then, the task of the module for postprocessing is just to merge these history files into a single history file of the whole model. It uses the timestamps of the events to

order them in the final result. Events that have the same timestamp, but are executed on different simulators are further ordered by their simulator ID.

5.4 Evaluation of the Implementation

This section concludes the description of the implementation of this thesis. First, I summarise the implementation and discuss its main advantages and disadvantages. Then, I conclude this section by an evaluation of the implementation, considering the goal of the thesis to achieve a positive speedup.

Implementation Summary The implementation of the parallel/distributed simulation system for wireless networks is an extension to the network simulator ns-2[9]. It has added the following new components:

- Network graph

  This component is responsible to represent the network model of ns-2 as a graph and partition it for the distributed simulation. It provides an interface to the `Metis` library[5].

- Simulator scheduler

  This is a simulator scheduler, that is aware of the parallel/distributed simulation, and uses a conservative algorithm to control its execution.

- Communication interface

  This module provides services to exchange messages and synchronise the execution of the model with other simulators in the system. It uses the `libSynk` library which implements most of these services[4].

In order to reconstruct the ns-2 simulator from a sequential to a distributed one, some of its components have been modified. These modifications have a management function, and do not change the semantics of the model. They are mainly in the following components:

- Wireless network

  These modifications are needed to keep the integrity of the network model across multiple simulators. At the points where the network is "cut" from the separation, the interactions between model components have to be replaced by exchange of messages among the simulators.

- Random number generation

  These modifications allow the replicated simulation of border nodes. They guarantee that multiple instances of border nodes have the same stochastic behaviour.

- Time module

  This is a structure that represents time in the simulation system. It is needed to ensure repeatability of distributed simulations.

The own implementation amounts around 4000, lines of code in C++ and OTcl. Three thirds from them are implementation of new components, and the rest are modifications of the ns-2 to manage the parallel/distributed simulation.

Advantages

- Modular Implementation

  The implementation followed the design and has also a modular nature. This allows to change some parts of the system without affecting other implementation. For example, the synchronisation algorithm can be replaced with another one which extracts higher lookaheads from the model and promises a higher speedup. This can be done without changing the partitioning, the algorithm of the scheduler, the operations for sending and receiving messages.

- Repeatability

  The implementation introduces composite timestamps to guarantee that distributed simulations are repeatable. The used method for repeatability is easy to implement and has a negligible cost. Composite timestamps were introduced in ns-2 without modifications of the simulation models, using object-oriented programming techniques.

Disadvantages

- Low-representative Graph Parameters

  The graph that represents the ns-2 network does not reflect the amount of computations that are needed to simulate a network node. The used models in the context of the thesis use communication protocols, where nodes have different roles and need different amount of processing power respectively. The roles of the network nodes are determined at run-time, and therefore are not considered by the partitioning, which occurs before simulation. If the amount of processing power was considered by the partitioning this would result in a better balance.

An alternative approach would be to start a sequential simulation for awhile, let the nodes determine their roles, then partition the simulation and continue it in a distributed way. This approach would also lead to a better estimation of the communication patterns between network nodes. This approach would also be a next step in moving towards mobility in distributed simulations.

- Small Lookaheads

  The lookaheads, that are used for distributed simulation imply very short safe windows and does not promise a parallelism. Even though the model provides knowledge to extract greater lookaheads, they can not be used at the current stage with the `libSynk` middleware.

Evaluation The solution of this thesis task provides a parallel and distributed simulator for wireless ad-hoc networks. I have solved the task to transform the simulator ns-2 from a sequential one to a parallel/distributed one. A distributed simulation now produces the same results as a sequential one. The following problems are still left to be solved in order to achieve speedup in simulation of wireless networks:

- Dynamic Lookahead

  A method for dynamically changing the lookahead in distributed simulation is needed in order to achieve a positive speedup. This step can be implemented as a part of the `LibSynk` library [4], which requires a very good understanding of its operation.

- Dynamic Repartitioning

  Dynamic repartitioning of the simulation model can enable mobility in distributed simulations of wireless networks. The main challenge here is the migration of nodes in the simulation model from one simulator to another.

A solution of these two problems will lead to a fully functional parallel and distributed simulator for mobile ad-hoc networks.

6 Experimental Evaluation

This chapter describes experiments, that verify and evaluate the implemented parallel and distributed simulation system. First, in 6.1 it presents the goals of the experimental study and expectations for the results, based on the design and the implementation. Then, section 6.2 describes the strategy for realisation of the experiments, and the structure of their description. Sections 6.3 and 6.4 describe the realisation of the experiments and present the results. Section 6.3 groups experiments that test empirically the correct operation of the distributed simulation system. Section 6.4 presents experiments that measure the speedup, gained from parallel simulation, and estimate the possible speedup from parallel and distributed simulation. And finally section 6.5 summarises the results of the experimental study, and evaluates their contributions to achieve its goals. It concludes this section with directions for further experimental studies.

6.1 Goals and Expectations of Experiments

Goals The experimental study has the following goals:

1. Prove correct operation

   The most important requirement to any computer program is that it operates as it is thought to. Therefore, the first goal of the experiments is to prove empirically the correct operation of the distributed simulation system, developed in this thesis. The proof here is only empirical, because a distributed simulation system is a highly complex system and it can be hardly analytically tested.

2. Show dependency from lookahead

   A significant drawback of the implementation of this thesis is the use of short lookaheads, which do not promise a positive speedup. Therefore, the second goal of the experimental study is to show that if the lookahead was higher, the speedup would be positive.

Expectations Based on the design and the implementation of this thesis and on preliminary test simulation runs, I can make the following expectations for the results from the experiments:

1. Distributed simulations operate correctly

   Distributed simulation does not introduce any additional model-related computations, because of the redundant simulation of border nodes. It allows to use methods of the sequential simulator to compute interactions between objects

in a distributed simulator. For example when a node sends a packet which needs to be synchronised, all nodes that may receive the packet are available in the same simulator. Then the model-related parameters of the transmission like reception power, propagation delay are computed locally, using the methods of the sequential simulator. This results in scheduling of one event for each receiver node in a sensing range. And if one of the receiver nodes is a border node, the corresponding event has just to be transmitted through a network and scheduled remotely. Therefore, I expect that distributed simulation does not change the semantics of the model in any way, and it produces the same results.

Another source of confidence is the use of the `libSynk` library for message exchange and synchronisation[4]. It has proved its correctness in other projects, and provides a correct interface for these technical aspects of parallel and distributed simulation.

2. Higher lookahead results in a higher speedup

   A higher lookahead in a conservative simulation results in longer safe windows, and therefore possibly more events, that can be executed in parallel. Therefore, I am sure that the use of the higher but dynamic lookahead, available in the model, would result in a higher speedup.

6.2 Organisation of Experiments

The experiments in this study are organised in the following manner:

Purpose of Experiment This is the first part of each experiment. It states what is the experiment trying to achieve, in the context of the goals of the experimental study.

Experimental Task The second part of an experiment describes which tasks are be done in order to achieve the goal of the experiment.

Expected Results In this part, I give my personal expectations about the results of an experiment. They are based on my view of the developed parallel/distributed simulation system and the context of each experiment.

Task Solution This part describes the solution of the experimental task, and the realisation of the experiment.

Discussion of Results Here I present, comment and explain the results of an experiment.

Parameter	Number of Nodes		Density $[nodes/km^2]$	
	Low	High	Low	High
Value	100	400	10	20

Table 6.1: Parameters for correctness test

Conclusion In the last part of an experiment, I evaluate the meaning of the results, and the contribution of the experiment to achieve the goals of the experimental study.

6.3 Correctness Tests

This section describes experiments that empirically test the correct operation of the developed distributed simulation system. The first experiment in 6.3.1 tests whether a distributed simulation produces the same results as a sequential one. And the second one in 6.3.2 tests whether distributed simulations are repeatable under changing network conditions.

6.3.1 Simulation Results Test

Purpose of Experiment The purpose of this experiment is to empirically prove that a distributed simulation produces the same results as a sequential one.

Experimental Task The task is to run network models in a sequential simulation, and in a distributed simulation on multiple computers and to compare the results. These models should be run with different parameters like number of nodes, and density. The results are files that contain information about events, happened during the simulation. These files have to be compared byte-wise in order to see every possible difference.

Expected Results Preliminary simulation tests show that distributed simulations produce the same results as sequential ones when using 2 simulators. Therefore, I also expect the same simulation results in simulations with more than 2 computers.

Task Solution For this distributed simulation I use a fixed number of 4 computers, which is chosen to be larger than a simpler case with 2 computers. However, the number is not higher than 4 because at the current stage there is no central controller of the distributed simulation, and simulators have to be started by hand. Then, for the number of nodes and density I choose 2 values respectively, one of them relatively low, and the other relatively high (see table 6.1).

Note that 20 $nodes/km^2$ is not a high density in a real wireless network, but a sufficiently high in a distributed simulation. This is because the sensing range in ns-2

```
< r 0.093951705 _24_ MAC  --- 1835008
---
> D 0.093951705 _24_ MAC  COL 1835008
```

Figure 6.1: Difference in simulation results

is 550m, which is relatively high for $1km^2$. This means that one node in the middle of a square area of 1000x1000m can affect almost all other 19 nodes in this area, and therefore a network with this density has a highly connected graph.

Different combinations of these parameters result in 4 different network models. On every node in these network models I run a broadcast application, that sends one packet in the network each second.

Discussion of Results The sequential and distributed simulation runs produce the same results in the cases with low network density. This means that the synchronisation method keeps the integrity of the model in this case. However, the results from distributed simulation with higher density differ from the sequential case. The differences show that some network packets are not delivered to their destinations, and therefore produce different behaviour of the corresponding network nodes. Figure 6.1 shows a part of the output from the `diff` program[2], that is used to compare the results byte-wise. The diff program compares the simulation results and suggests that the first line is replaced by the third line in order to make them the same. The first line is from the distributed simulation, and the third one from the sequential simulation. These lines show the action of the MAC layer at node number `24` at time `0.093951705`. Both nodes process the same packet at the same time, but behave in a different way. In the distributed simulation node `24` receives the packet with ID `1835008` (shown by the `r` sign). In the sequential simulation node `24` drops the packet, because of a collision at the MAC layer (shown by the `D` and `COL` signs). This result shows that the current packet with ID `1835008` is delivered correctly by the distributed simulation at the same time as in a sequential simulation. The different behaviour of the MAC layer can be caused from a previous packet, which was received in the sequential simulation, but not received in the distributed one.

A deeper look at the model shows that this effect comes from the used scheme for replicated execution of border nodes. This scheme assumes that when a border node sends a packet, it does not need to be synchronised with other simulators. But there are network scenarios that break this assumption. Figure 6.2 shows one of them. This is a model of a network, divided into three parts, and executed on three simulators. The figure shows the operation of the method for replicated borders. Simulator 1 has a connection to the foreign node B from part 2, and therefore maintains an instance of B. Similarly, simulator 2 maintains instances of A and C, and simulator 3 maintains

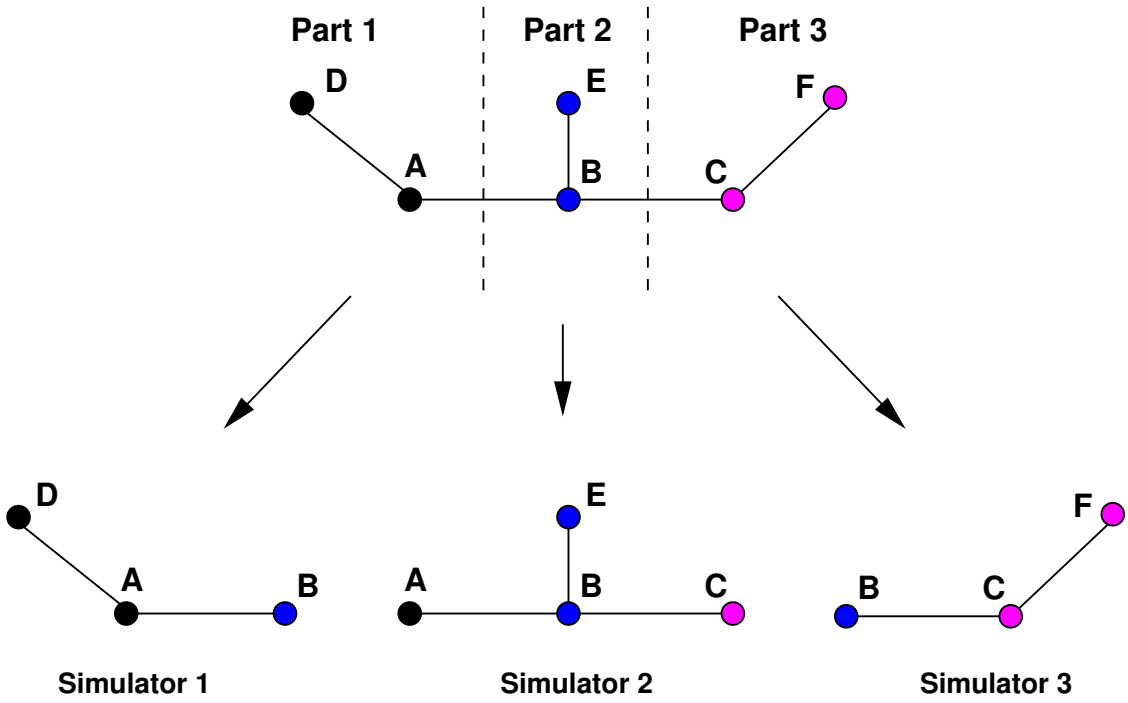


Figure 6.2: Problem of replicated simulation

an instance of node B. Now, the method for replicated execution of border regions marks node A as *border*, and when it sends a packet it does not synchronise it with other simulators. This is correct for simulator 2, because it is a neighbour to simulator 1, and also maintains an instance of A. But simulator 3 does not maintain node A, because its own node C does not have a direct connection to it. However, the transmission from node A affects node B, which is represented in all simulators. Since simulator 3 is not informed about this interaction its instance of node B has a different and wrong behaviour. Therefore, the simulation does not produce the same results as a sequential one.

This effect comes from the fact that the method for replicated execution considers only one possible role per node. The nodes A, B, and C have a role of *borders*, and the nodes D, E, and F a role of *disturber*. And this method initiates synchronisations only when a *disturber* node sends a packet. For instance, when node D sends a packet, simulator 1 schedules it locally for node A, and informs simulator 2 for the change of node A via a message. Now in order to operate correctly, this method needs to consider the roles of the nodes, depending on their neighbourhood. For simulator 2 node A is a *border*, and its activities do not need to be synchronised. But for simulator 3 node A is a *disturber*, and it needs to be synchronised.

This effect appears only in the model with higher density, because there the graph

has a higher connectivity, and introduces this special case. The simulation with low density is correct, because the above effect does not appear there.

Conclusion This experiment shows two important results. First, the distributed simulation produces the same results as a sequential, when the model does not contain some special situations. This result shows that distributed simulation of wireless networks in ns-2 is *possible*. It also proves that replacement of interactions in the sequential model with message exchange in the distributed model is technically correct, and leads to the same simulation results.

The second important conclusion is that the method for replicated execution of border regions has to be refined in order to be used in more complex network scenarios. And since all network scenarios can not be tested, this method has to be also mathematically proved in order to be used for large-scale network simulations.

6.3.2 Repeatability Test

This experiment tests the distributed simulation system under different network conditions and checks, whether it behaves in a deterministic way.

Purpose of Experiment Repeatability is an important feature of scientific experiments, including simulations. However, a deterministic behaviour of a distributed simulation system is a non-trivial issue. This is because simulators are separated systems, interacting through a communication medium. The main concern are *simultaneous messages* (also called concurrent messages), that are messages with the same simulation time. These messages may arrive in different order, depending on delays through the communication system, and if the order of their processing is not defined, they can lead to nondeterministic simulations. In this thesis I implemented a method to ensure a deterministic behaviour of the simulators in the distributed system, under nondeterministic network conditions. So, the goal of this experiment is to show that the developed distributed simulation system operates in a deterministic way, regardless of the changing network conditions.

Experimental Task The task of this experiment is first to find an appropriate model which can test the repeatability of the simulation system. Appropriate means a model that introduces messages with the same timestamps between simulators. Then, the second step is to run this model under different network conditions with different communication delays. During these simulation runs, an observer has to keep track of the execution of the simulation.

This task requires also simulation runs with the repeatability feature turned off, in order to see the possible different behaviour under different network conditions in this case.

Expected Results The technique for repeatability, used in this work is straightforward, and based on simple mathematical and logical rules. The idea is to request from the synchronisation layer a time advance ΔT after the time of the next simulation event T_e. In this case the synchronisation middleware delivers all possible messages with a timestamp less than $T_e + \Delta T$, which means also all messages with timestamp T_e. Therefore, all events with timestamp T_e are known and ordered based on their sender ID, before the first event in the row is executed. Since this technique is simple and easy to implement, I expect a repeatable operation of the simulation system.

Task Solution The simplest scenario to test the repeatability are two simulators, which send messages with the same timestamp to a third one. In order to achieve this behaviour, I create the following network model (see figure 6.3). The model is divided into three parts, where nodes 0 and 1 belong to partition 1, nodes 2 and 3 to partition number 2, and the nodes 4 and 5 to the third partition. Each network partition is assigned to a separate simulator. Since the method of the border nodes is used, the neighbouring nodes in the middle 1, 2, and 4 are simulated by all simulators. Then, the simulators exchange messages when the nodes 0, 3, and 5 send packets in the network. In order to achieve messages at the same timestamp, the distances between the nodes, $0 \rightarrow 1$, $3 \rightarrow 2$, and $5 \rightarrow 4$ are the same. These equal distances guarantee the same propagation delays and if the nodes 0, 3, and 5 send packets in the same simulation time, this will result in three messages with the same timestamp. In order to incite these three nodes to send into the network at the same time, I start an application on each node, which sends one packet per second, starting at time 0 (0, 1, 2, ...). This application behaviour will not result in simultaneous messages each second, because of the random backoff access method to the wireless medium. But when this model runs long enough (100000 seconds for example), it is very likely that these simultaneous messages occur.

This model is then run in a distributed simulation on three computers, connected by a Fast Ethernet network. In order to introduce different network conditions, I use a special queueing discipline on the network interface of simulator 1. This is a Token Bucket Filter (TBF) queueing discipline with maximum data rate of 10MBit/s, and latency of 100ms[8]. In a normal operation simulator 1 sends data into the Ethernet network with a speed of around 13MBit/s. This queueing discipline slows down the sending speed and introduces an additional latency which can delay messages during simulation. Simulator number 1 has the smallest ID, and therefore the highest priority messages, when they contain the same timestamp. This queueing discipline delays messages, coming from simulator 1, and increases the probability for out of order processing of simultaneous events.

During the simulations an observer tracks messages, in the order which they are

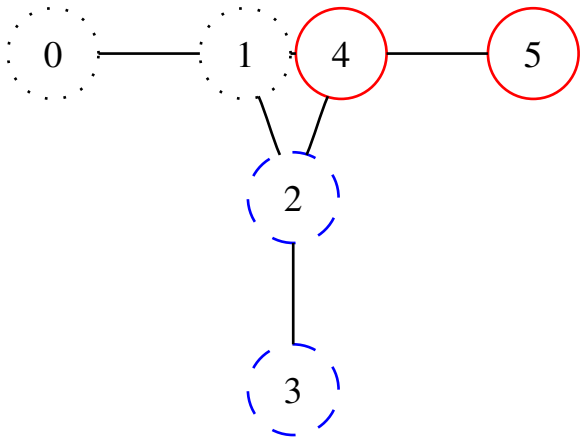


Figure 6.3: Network model for repeatability test

	FIFO order			Deterministic order		
	Sim1	Sim2	Sim3	Sim1	Sim2	Sim3
Out of order messages [%]	2.3	4.9	2.1	10.8	89.6	11.3
Out of order external events [%]	2.3	4.9	2.1	0	0	0

Table 6.2: Repeatability test — symmetric network

received. Additionally, it follows the order of execution of the corresponding *external events*. The goal of this monitoring is to see whether out of order messages result in out of order events.

These simulations are also run with the repeatability feature turned off in order to see the different behaviour by a pure FIFO ordering of simultaneous messages.

Discussion of Results The results from this experiment are shown in tables 6.2 and 6.3. Table 6.2 shows results from an experiment under normal conditions, i.e. a symmetric network. Table 6.3 presents results from the asymmetric network, i.e. the network with a queueing discipline on simulator 1. In each network I run a simulation with a FIFO ordering, and a simulation with deterministic ordering of simultaneous events. Then, for each simulator I measure the percentage of all *concurrent messages*, that it receives out of order. Each external message in this scenario results in one external event in the simulator queue. Therefore, I also measure the percentage of all *concurrent external events*, that are executed *out of order* according to the defined ordering notation. This last measure shows whether the simulation is deterministic or not.

As expected, the communication system does not follow the semantics for order of simultaneous messages, defined in the simulation system. The results show that

	FIFO order			Deterministic order		
	Sim1	Sim2	Sim3	Sim1	Sim2	Sim3
Out of order messages [%]	3.8	98.4	98.6	3.6	98.4	98.8
Out of order external events [%]	3.8	98.4	98.6	0	0	0

Table 6.3: Repeatability test — asymmetric network

even a symmetric network introduces out of order messages (see table 6.2). Moreover, the percentage of out of order messages varies by the different simulation runs, which shows that the order of messages is nondeterministic. All these messages result in out of order execution of events, by the FIFO ordering. But the deterministic ordering manages the unordered messages, and executes all external events in order.

The asymmetry in the network in the second series of simulations results in more messages, received out of order (see table 6.3). This is because the highest priority messages from simulator 1 are now delayed by the network. Therefore, the other two simulators receive almost all of the concurrent messages out of order. The FIFO ordering again keeps the order of the messages as it is, and executes almost all concurrent external events out of order. But the deterministic ordering is aware of the fact that messages can be delayed through the communication system. In spite of the high proportion of unordered messages, it executes all external events in a repeatable fashion.

Conclusion This experiment shows that the simulation system executes all external concurrent events in a deterministic order. The other events, which are not concurrent also have their strict order, defined by the synchronisation algorithm. This means that all events in each simulator are executed in the same order from one execution to another. Therefore, assuming that event computations are repeatable, the distributed simulation system produces repeatable results, which is another aspect of its correctness.

6.4 Speedup Tests

This section groups experiments that test the speedup from parallel simulations. Section 6.4.1 presents an experiment that measures speedup, and the influence of lookahead in a parallel simulation. Section 6.4.2 estimates the possible speedup from parallel simulation, if dynamic lookahead was used.

6.4.1 Speedup Measurement

This section describes an experiment that measures speedup in parallel simulation.

Purpose of Experiment Since the lookaheads, used in the current implementation of the simulation system are small, it can hardly achieve a positive speedup. Therefore, the purpose of this experiment is to show that a higher *speedup* is possible, if the lookaheads were higher.

Experimental Task In order to achieve the goal of the experiment, the following tasks have to be done:

1. Construct different simulation models, which provide different lookaheads. One of these models should provide a maximum lookahead, possible from this parallel/distributed simulation system.

2. Run each model in a sequential, and in a parallel/distributed simulator, and compute the speedup for each model.

3. Compare the different speedups and estimate the possible speedup, if the lookahead was higher.

Expected Results The lookaheads, in the current implementation are based on propagation delays, and therefore are very small ($[1...1500ns]$). This results in short safe windows, and smaller number of events which a simulator executes during its safe window. The shorter the safe window, the more often the simulators have to stop event processing and do synchronisation. Therefore, I expect that a simulation with these lookaheads will result in a negative speedup.

On the other hand, special cases exist, where the lookahead can be set to infinity. This is for example when two subnetworks are situated at a very big distance and do not affect each other at all. In these cases I expect a maximum positive speedup from the simulation, because there are practically no synchronisations.

Task Solution In order to have a control over the lookahead, I choose to simulate two subnetworks, with a variable distance between them. Since the lookahead is determined by the minimum propagation delay between these subnetworks, a higher distance results in a higher lookahead. And when the distance is more than 550m, which is the sensing range in the ns-2, the two subnetworks are completely independent and then the lookahead can be set to infinity.

So, I generate different models of networks, which consist of two subnetworks with a variable distance between them (100m, 200m, 400m, and 600m). I also use different numbers of network nodes for these models (100, 400, 700, and 1000). This can give a feeling, how the number of nodes in the network influences the speedup.

These models were run in a parallel simulation on a two-processor Pentium III machine with a frequency of 450MHz.

Discussion of Results Figure 6.4 shows the results from this experiment. The first result to be noticed is that the models with small lookaheads [$300ns$...$1200ns$] achieve a negative speedup. As expected, this comes from short safe windows and small number of events, executed in parallel. Even when the lookahead is increased 4 times from 300ns to 1200ns, meaning possibly 4 times more events processed in parallel, the speedup increases only very few.

Lets consider two different test runs. One is by lookahead 300ns, and 100 nodes, and the other is at lookahead 1200ns and 1000 nodes. The second test has 10 times more nodes, which implies around 10 times more events within the simulation, so their distribution in the time scale is 10 times more frequent. The second test has also a 4 times larger lookahead, which implies around 4 times larger safe window. These mean that during the second simulation run there were around 40 times more events per safe window than in the first one. Nevertheless, the increase in the speedup is only 0.1. This means that the work, done on synchronisation takes much more time than the work, done on processing of events. Therefore, these results induce another interesting question. How many times more events per safe window are needed in order to increase the speedup with 1, and make it in the interval [1.4...1.5]? This might be a question for an additional experimental study.

Another result from this experiment is that when the lookahead is infinity the speedup is positive. This result gives the maximum possible speedup, that can be achieved. Theoretically the speedup in this case is equal to the number of processors, in this case 2, because they do not interact during the simulation. But in practice there are factors that reduce this speedup. In this implementation all simulators start simulation with the whole model. Each simulator interprets and separates it on its own, and then activates only one part of it, which it then simulates. This work is done from both simulators, so it does not contribute to parallel processing and to the speedup. But the results show that when the pure simulation work increases 10 times from 100 to 1000 nodes, it compensates this duplicated work at the beginning and increases the speedup with around 0.5.

Conclusion This experiment gives an estimation of the maximum possible speedup from a parallel simulation. It shows that this maximum speedup is positive and close to the theoretical maximum. So, it shows that a speedup with this simulation system is *possible*, if the lookahead was higher. However, this experiment does not answer how longer the lookahead needs to be in order to achieve a positive speedup. Also how much speedup is possible with the dynamic lookahead in the simulation model?

6.4.2 Speedup Estimation

This section presents an experiment that *estimates* the possible speedup from parallel simulation, if the lookahead was higher.

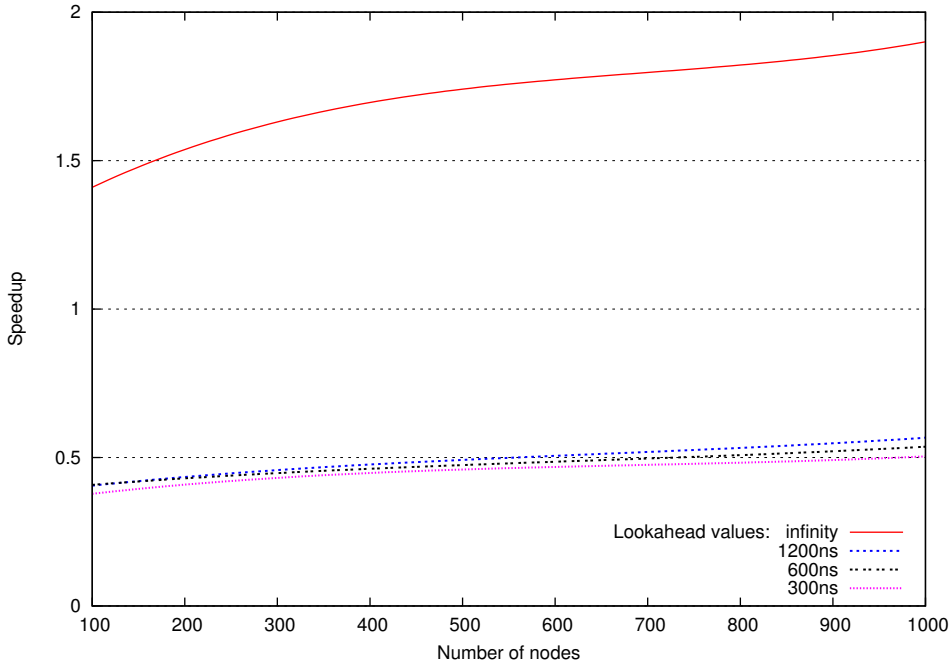


Figure 6.4: Speedup depending on lookahead

Purpose of Experiment The purpose of this experiment is to answer the questions, that the previous experiment brought up:

1. How much lookahead is needed in order to achieve a positive speedup in the non-trivial case when lookahead is not infinity?

2. Is this lookahead available in the simulation model?

Experimental Task The first task of this experiment is to *artificially increase* the lookahead in wireless network simulations with ns-2, and to measure the resulting speedup. The lookahead is to be increased until it results in a positive speedup. Then, the second task is to estimate the possible dynamic lookahead, available in the simulation model.

Expected Results The previous experiment showed that a short lookahead results in a speedup of around 0.5, and a lookahead of infinity in around 1.7. Therefore, I expect that there is a value of the lookahead that causes the speedup to be a positive number around 1. However, this value can be hardly estimated beforehand.

For the possible dynamic lookahead, I expect values in the range $[1...30ms]$, because of the operation of wireless networks in the context of this thesis (see section 2.5).

Task Solution The currently used lookahead in parallel and distributed simulations is fixed and based on propagation delay. Therefore, the only way to *artificially increase* the lookahead is to change the propagation delay. The propagation delay between two network nodes A and B in ns-2 is calculated by the formula:

$$Propagation\,Delay = \frac{Distance(A,B)}{Speed\,of\,Light}$$

Therefore, in order to increase the propagation delay one needs either to increase the distance between nodes, or to decrease the speed of light in the model. Since the maximum possible distance ($550m$) still results in a very small propagation delay ($1.8\mu s$), the only way to further increase the propagation delay is to *decrease the speed of light in the model.* For this experiment I choose a speed of light of $100km/s$, which results in a propagation delay of $1ms$ at a distance of $100m$. The network topology is again two subnetworks with a variable distance between them, like in the previous experiment. The network nodes run a real-time communication protocol, used in the context of this thesis (see section 2.5). The experiments start with a distance of $100m$.

The possible dynamic lookahead is theoretically estimated in the range $[1...30ms]$, but it depends on the dynamics of the network. In order to estimate in more precisely I measure the time intervals in simulation time in which the simulators send messages to each other. These time intervals depend on the behaviour on the nodes in the border areas of the network.

Discussion of Results Figure 6.5 shows the speedup of this simulation with a distance between the networks $100m$, and a lookahead of $1ms$. This speedup is clearly positive, which means that a lookahead of $1ms$ is enough to achieve a positive speedup in parallel simulation. Figure 6.6 shows the other result from this experiment. It is a distribution of the time intervals in simulation time between two consecutive messages send by the LPs in this parallel simulation. The graphic shows that these intervals are in nearly 90% of the cases in the range $[0...30ms]$. This a rough estimate of the dynamic lookahead, available in the simulation model, because it is a result from one simulation, and not from analytical evaluation of the model. In order to be used in a conservative simulation, these lookaheads have to be analytically proved. It is possible that only parts of this dynamic lookahead can be proved and used for synchronisation. But nevertheless, the magnitude of the available lookahead is higher than the lookahead, needed for a positive speedup in this case. Therefore, it gives a good promise for a positive speedup even in distributed simulations, where synchronisation takes more time.

Conclusion This experiment shows that the lookahead, needed for a positive speedup in a parallel simulation, is available in the simulation model. The available lookahead is

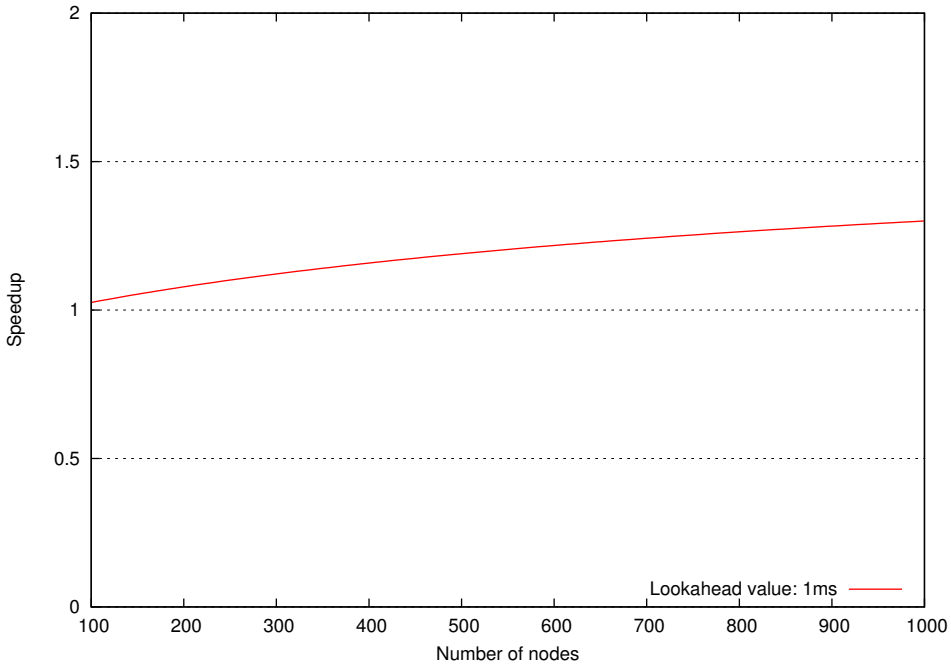


Figure 6.5: Estimated speedup

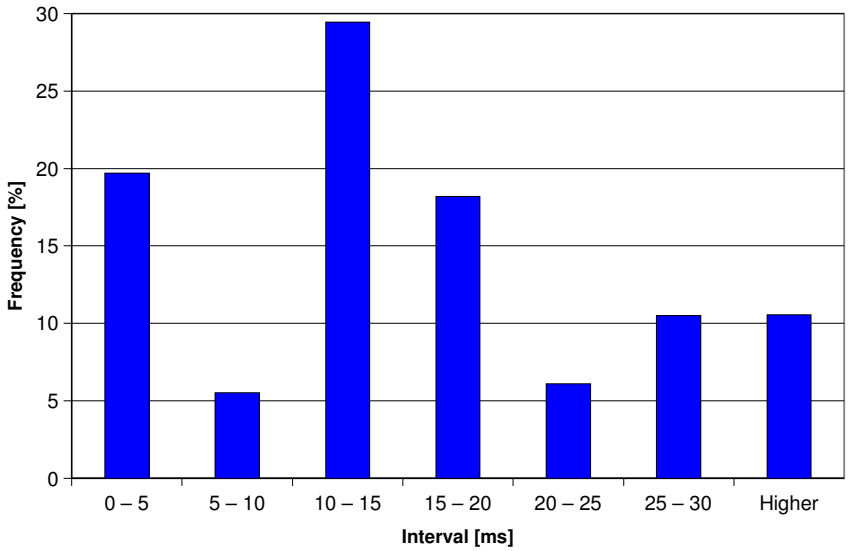


Figure 6.6: Message intervals distribution

even with magnitudes higher, which also gives a promise for a speedup in a distributed simulation. These are important conclusions, that contribute to the goal of this thesis. They show that if additional efforts extract a higher lookahead, a *positive* speedup of wireless network simulations is *possible*.

6.5 Interpretation of Results

This section evaluates the achievements of the experimental results in the context of the goals of the experiments. Then it gives possible directions for future experimental tests.

Evaluation of Results These experiments showed that distributed simulation correctly translates interactions between parts of a sequential model into interactions between simulators in a distributed model. The used technique for replicated simulation of border regions keeps the model-related parameters of these interactions correct, and results in correct distributed simulations. However, some special situations in the network models show that this technique is not completely precise in informing all interested simulators about an interaction in the model. The technique for replicated simulations needs to be additionally revised in order to produce correct results in all cases.

Another contribution to the correctness of distributed simulations is the proof of their repeatability. The experiments showed that distributed simulation operates in a deterministic way under a nondeterministic behaviour of the communication system. This means that distributed simulations produce the same results from one run to another, and can be used to correctly test and follow errors in a simulation model during its development.

The experiments also showed that the use of short lookaheads results in a negative speedup, even in parallel simulations, where synchronisation is much faster than in distributed simulations. This means that additional model-related information has to be used in order to achieve longer lookaheads and a higher speedup. Furthermore, the experiments showed that the use of this information would result in a positive speedup in parallel simulation.

Future Experimental Tests A question for future experimental tests is the speedup in distributed simulation on multiple computers within one network. These experiments require the development of a testing environment, and a central controller to maintain the execution of experiments and gather the results.

An Interesting experimental question is for example the dependence of speedup from different factors in the network model. These can be density of the network, structure of the network (clustered or uniformly distributed nodes). Another factor that might

influence speedup is the behaviour of the applications, i.e. the part of intra-cluster and inter-cluster communications. Another practically related parameter is the number of processors in a distributed simulation. It would be interesting to see how the number of processors influence the speedup, and what is the optimal number of processors for a given simulation scenario. These experiments require the use of dynamic lookahead in order to promise a positive speedup.

7 Conclusions and Outlook

This final section summarises the main aspects of the thesis, the lessons from it and gives directions for future research.

Summary This thesis work is motivated by the time-consuming simulations of one particular network simulator — the ns-2[9]. These long-running simulations delay the development cycle of communication protocols for wireless networks. Therefore, the goal of this thesis work is to reduce the running time of wireless network simulations with ns-2 by using multiple processing units to complete a single simulation task. The task of the thesis, arising from this goal, is to extend the ns-2 to run simulations on multiple computers within one network. The main challenge of this task is to keep the entirety and correct execution of the simulation model during a distributed simulation. Due to the large scale of the used models (100...1000 nodes), a necessary preprocessing step is to automatically divide the network model into multiple parts, suitable for a distributed execution.

Within this thesis I have designed and implemented a distributed simulation framework in the network simulator ns-2. It allows to automatically separate the simulation model, using a graph partitioning technique. The used algorithm[26] is specially optimised for distributed simulations, and implemented by the Metis graph partitioning library[5]. This method for network separation automatically divides the network into groups of nodes (partitions), and assigns each partition to a separate simulator. In a sequential simulation these partitions interact by exchanging events. In order to keep the integrity in a distributed simulation, the simulators exchange messages through a network to inform each other about these interactions. These messages are exchanged using the communication and synchronisation middleware LibSynk[4]. It provides a high level abstraction to message exchange and synchronisation in distributed applications, and can operate both over a computer network and over a shared memory. This provides a flexibility to run both parallel and distributed simulations with the *same* implementation. On top of the synchronisation middleware, I have designed and implemented a module to guarantee repeatability of distributed simulations, regardless of the properties of the communication system. This method is effective, easy to understand, and has a reasonable cost. Experimental results show that the parallel and distributed simulation system operates correctly, and produces the same results as a sequential simulation. It has also shown its deterministic behaviour under changing network conditions, which is a proof for repeatable simulations. This is a solution to the task of the thesis.

However, the solution of the task did not suffice to achieve the goal of the thesis, i.e. a positive speedup. This is because the used technique for synchronisation in distributed simulation utilises a low part of the parallelism, available in the simulation model. This results in a high amount of work by the management of

the distributed simulation. In fact the management work takes even more time than the pure simulation work, because it also includes communications through a network. Therefore, distributed simulations are also time-consuming and even slower that sequential simulations.

But nevertheless, the solution of the thesis task has a contribution to achieve the goal. This is a basic system for parallel and distributed simulation, which has proved its correct operation. Moreover, experimental results show that a positive speedup is possible with the use of a higher level knowledge about the simulation model. This knowledge is available, and results in a positive speedup in parallel simulations. Furthermore, it gives a reasonable promise for a positive speedup in distributed simulations of wireless networks. In this sense the solution of the task of this thesis is a correct step for the achievement of its goal.

Outlook The result of this thesis work is a parallel and distributed simulation system for wireless networks. At this stage it can be used to speedup simulations of stationary wireless networks, containing independent subnetworks. These independent subnetworks have to be either at a big distance, or to use different wireless communication channels in order to avoid the need of synchronisation, which is the current bottleneck of the distributed simulation.

The developed simulation system can be also used as a base for further development and improvements. They are needed in order to achieve the goal of this project. There are two main aspects which have to be further investigated.

A first aspect for the near future is adaptive synchronisation. It considers the use of more information from the model in order to reduce the work for management of distributed simulation, and to increase the speedup. This information is available in the model, but has a dynamic nature. It requires a method to dynamically change the lookahead in the distributed simulation system.

A challenge for the far future is distributed simulation of mobile networks. When the network nodes in the model move they also introduce dynamics in the simulation infrastructure. In this case the simulation framework has to track the interactions and changes occurring in the model. In order to keep its integrity it has to dynamically decide which other simulators in the system to inform about these interactions. Furthermore, when the network nodes move, they might scatter from their initial position and do not form a cluster suitable for a distributed simulation. In these cases it is possible that a repartitioning of the model is needed in order to achieve higher speedup. So, distributed simulations of mobile networks need to be periodically reorganised, and repartitioned in order to achieve a good performance. The main challenge here is not the repartitioning itself, but the migration of network nodes in the model from one simulator to another. Since the ns-2 is a sequential simulator such migration possibilities are not planed in its design. Moreover, it is optimised for

a sequential simulation, and network nodes are tightly coupled with the simulation framework for a best sequential performance. Therefore mobility is another big challenge for distributed simulation with ns-2.

When further investigations solve these problems, the result of this thesis can be used to speedup wireless network simulations. This can result in a simulation framework, that helps to increase the quality of communication protocols, and satisfy the increasing requirements for mobile communications.

Lessons This thesis has also positive contributions to my personal development. These are gains both in the area of computer science and in personal aspect.

The most significant gain to my professional competences are the knowledge and experience in the area of distributed simulation. I had to cope with typical problems in distributed systems like serialisation/deserialisation of data structures and unpredictability of communication flows. Furthermore, the study and use of parallel algorithms developed my abilities to think in parallel on behalf of multiple processing units, participation in one operation. I am sure that these gains will contribute to my further development in the field of network technology, where similar problems arise.

The personal competences have even a higher value for me than the professional ones. This is because professional skills have a pure technical nature. They are something transitional, and their relevance may fade with time. This is because problems that we cope with now might not be real problems of the future. While the personal skills build a maturity, which is more general and will be always needed.

The most important lesson for me in personal aspect is to thoroughly brainstorm and prove new ideas, before starting to use them. Sometimes we have "such a great new idea" that we believe that it brings to the desired result, only based on a cursory judgement. One has to be sure that the new "great idea" brings to the wanted result, before starting to use it. It is even better to test new ideas with other people, which can have an objective view of our thinking. Even though this lesson came from a new idea for the design of the simulation system, it has a general contribution. New ideas will continue to come in the future, no matter of their context or the state of the technology, and they have to be treated appropriately.

References

[1] AutoPart: simulation partitioning tool for PDNS. http://www.cc.gatech.edu/grads/x/Donghua.Xu/autopart/.

[2] Diffutils: tools to comparing and merging files. http://www.gnu.org/software/diffutils/diffutils.html.

[3] Eine Publisher/Subscriber-basierte Middleware mit Dienstgüte-Garantien zur Unterstützung kooperativer Anwendungen. http://www-ivs.cs.uni-magdeburg.de/EuK/forschung/projekte/spp1140/index.shtml.

[4] libSynk: library for communication and synchronisation in distributed applications. http://www.cc.gatech.edu/fac/kalyan/libsynk.htm.

[5] METIS: family of multilevel partitioning algorithms. http://www-users.cs.umn.edu/ karypis/metis/index.html.

[6] Nam: Network Animator. http://www.isi.edu/nsnam/nam/.

[7] PDNS: Parallel/Distributed NS. http://www.cc.gatech.edu/computing/compass/pdns/.

[8] TBF: Token Bucket Filter queueing discipline. http://www.die.net/doc/linux/man/man8/tc-tbf.8.html.

[9] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/.

[10] The WINDECT Website. http://www.windect.ethz.ch/.

[11] Luciano Bononi et al. A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems. In *IEEE International Symposium on Distributed Simulation and Real-Time Applications*, Eighth, pages 178–187. IEEE Computer Society, 10 2004.

[12] K. M. Chandy and J. Mirsa. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, 5(5):440–452, 1978.

[13] Farid Dowla. *Handbook of RF & Wireless Technologies.* Elsevier, Inc., 2004.

[14] Jerry Banks et al. *Discrete-Event System Simulation.* Prentice Hall, second edition, 1996.

[15] Alois Ferscha. Parallel and distributed simulation of discrete event systems. 1996.

[16] G. Fettweis, T. Hentschel, and E. Zimmermann. WIGWAM - a wireless gigabit system with advanced multimedia support. In *VDE-Kongress*, 2004.

[17] H. T. Friis. A note on a simple transmission formula. *Proc. IRE*, 34, 1946.

[18] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems.* Wiley, 2000.

[19] Jim Geier. *Wireless LANs: Implementing Interoperable Networks.* Macmillan Network Architecture & Development Series, 1999.

[20] Andre Herms and Daniel Mahrenholz. Unified Development and Deployment of Network Protocols. In *Proceedings of MeshNets*, 2005.

[21] Graham Horton. Introduction to Simulation. Lecture in Computer Science, 2004.

[22] Institute of Electrical and Electronics Engineers (IEEE), Inc. *ANSI/IEEE Std 802.11, 1999 Edition*, 1999.

[23] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[24] Z. Ji et al. Optimizing parallel execution of detailed wireless network simulation. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS)*, 2004.

[25] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, University of Minesota, 1995.

[26] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, University of Minesota, 1995.

[27] George Karypis Kirk Schloegel and Vipin Kumar. Graph partitioning for high performance scientific simulations, 2000.

[28] Kiran Madnani and Boreslaw K. Szymanski. Integrating distributed wireless simulation into genesis framework. In *Proc. Summer Computer Simulation Conference*, Montreal, Canada, 2003.

[29] T. S. Rappaport. *Wireless Communications, Principles and Practice.* Prentice Hall, 1996.

[30] Stefan Schemmer. *A Middleware for Cooperating Mobile Embedded Systems.* PhD Thesis, Otto-von-Guericke-University, 2004.

[31] F.A. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple Access and Busy Tone Solution. In *IEEE Trans. on Commun.*, 1975.

[32] Wave7 Optics, Inc. *TCP Performance on Broadband Networks*, 2002.

[33] XTP Forum. *Xpress Transport Protocol Specification*, 1998.

[34] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *12th Workshop on Parallel and Distributed Simulation*, 1998.

Declaration of Independence

I declare that I have developed this master's thesis independently, and only with the use of the specified sources.

Svilen Ivanov
Magdeburg, the 20.06.2005