RoboNetSim: An Integrated Framework for Multi-robot and Network Simulation

Michal Kudelski, Luca M. Gambardella, Gianni A. Di Caro[1]

Dalle Molle Institute for Artificial Intelligence (IDSIA) - Lugano, Switzerland

Abstract

In networked multi-robot systems, communication plays a major role defining system's dynamics and performance. Unfortunately, existing multi-robot simulators do not provide advanced communication models. Therefore, given the intrinsic unreliability of wireless communications, significant differences might be observed between simulation and real-world results.

Addressing these issues, we present *RoboNetSim*, an integrated simulation framework for communication-realistic simulation of networked multi-robot systems. RoboNetSim's integrates multi-robot simulators with network simulators. We present two model implementations based on ARGoS at the robotic side, and NS-2 and NS-3 as network simulators. We evaluate the framework in terms of accuracy and computational performance, showing that it can efficiently simulate systems consisting of hundreds of robots.

Using the Stage simulator as an example, we also show the integration of a robotic simulator with RoboNetSim by only adapting robot controllers, without the need to adapt the general code of the simulator.

Finally, we demonstrate the effects of communication on mobile multi-robot systems. We consider two different case studies: a distributed coordination and task assignment scenario, and a coordinated mobility scenario. We compare realistic network simulation with simplified communication models and algorithms, and we study the resulting behavior and performance of the multi-robot system and the impact of different parameters.

Keywords: Networked robotics, simulation, communication, distributed coordination, multi-robot systems

1. Introduction

There are three main motivations behind the use of multi-robot systems compared to single-robot approaches. The first one is robustness, which is particularly important in autonomous robotics: if one of the robots fails, other robots can potentially take over its tasks. The second motivation is efficiency: some tasks can be naturally solved more efficiently by exploiting the intrinsic parallelism and distributedness of multi-robot systems. The third one is efficacy: tasks that may be unreachable for individual robots could be effectively solved by a robot team through an explicit synergy of behaviors. These motivations lie at the roots of swarm robotics [1], and other approaches to distributed robotics.

Although the robots of a multi-robot system can be designed to act as autonomous and independent agents, in most cases the key to best exploit the capabilities of a multi-robot system lies in the use of *information sharing*. Coordination, cooperation and behavioral synergies, they all require the robots to exchange information. This applies to autonomous robotic systems, where robots exchange the information within a team, as well as to externally controlled systems, where robots may be required to aggregate gathered information and send it to the control center to close the loop. In any case, effective communication tools and techniques are desired in order to unleash the full power of a multi-robot system.

Robot communication can be realized in many different ways using different physical interfaces. In the literature one can find approaches based on *indirect communication* via sensing or via interaction with the environment (see [2, 3] for a review), or based on *simple direct communication* between robots, e.g. utilizing visual communications based on LEDs [4]. If robots are not mobile, fixed telecommunication lines can be reliability employed for high-bandwidth data exchange. However, in the context of mobile multi-robot systems, which is the case we focus on in this paper, it is rather obvious that *wireless radio communication* is the most powerful technique in terms of fast and effective information sharing (e.g. [5]). However, when using wireless communications, one basic problem arises. On the one hand, we expect the communication tools to be effective and trustworthy in order to enable the correct implementation of desired cooperation and coordination strategies. On the other hand, in the real world communications are not fully reliable. This is particularly true for *wireless mobile ad hoc networks* (MANETs [6]), which are the natural choice for fully autonomous multi-robot systems. The presence of unreliable communication can seriously affect the actual performance of a multi-robot system. Therefore, this issue needs to be taken into account while designing and simulating such systems.

In multi-robot (or multi-agent) systems, one typical approach for dealing with the problem of unreliable communication consists in trying to avoid or limit communication as much as pos-

Email addresses: `michal@idsia.ch` (Michal Kudelski), `luca@idsia.ch` (Luca M. Gambardella), `gianni@idsia.ch` (Gianni A. Di Caro)

[1]Gianni A. Di Caro is corresponding author.

sible, emphasizing the pure locality of interactions. This approach is popular in swarm robotics, where, for instance, the above mentioned indirect communication is commonly used. This way of proceeding, although it may increase the robustness of the system as a whole, does not allow to fully benefit from effective information dissemination and long-range interactions that would be available from advanced wireless telecommunication techniques. In simulation-based works, another typical approach is to make idealized assumptions about communication. For instance, in [7] authors assume that messages sent by robots via broadcasting are immediately available to all other robots. This approach allows to focus on other scientific issues, yet makes the study less realistic. In fact, we claim that, in order to reflect what happens in the real-world and properly validate high-level team strategies, a communication model for networked robotic systems, especially for large-scale ones, needs to consider the operations of different network protocols, radio interferences, and MAC layer collisions. Thus, a reliable and detailed (packet-level) communication model is required. As a matter of fact, only a few works deal with the problem of realistic simulation of communication among networked robots (see Section 2.3).

The above observations led us to the conclusion that there is a strong need for investigating the actual influence of communication on the operation of multi-robot systems. Such studies can be performed either on real robots or using advanced simulation tools. Unfortunately, real testbeds are costly, difficult to realize and often limited in the number (and/or size) of experiments that can be done. At the same time, there exists no single tool that offers advanced and realistic simulation of both robotic and communication issues.

The aim of this paper is precisely to propose a simulation tool which is communication-realistic and helps understanding the relation between the quality of communication and the operation of a large multi-robot system (a partial version of this work was published in [8]). We propose *RoboNetSim*: a general framework that is capable of combining physics-based multi-robotic simulators with state-of-the-art network simulation tools. We apply the proposed architecture and implement two integrated simulation environments, both based on open source tools, namely the *NS-2* [9] and *NS-3* [10] network simulators and the *ARGoS* simulator, designed for swarm robotics [11, 12]. The first environment combines ARGoS with NS-2, and the second one combines ARGoS with NS-3. We perform an extensive experimental study to evaluate the general framework architecture and the instantiated simulation tools in terms of computational performance, correctness, and generated overhead. We show that the integrated framework is able to efficiently and realistically simulate systems consisting of hundreds of mobile robots.

The two integrated environments above are based on the full integration of a robotic simulator with a network simulator: they provide additional modules of the robotic simulator (e.g., sensors, actuators, etc.) that make the integration with a network simulator transparent to the user who develops robot controllers. Since this integration process requires some structural changes in the simulators at hand, we also demonstrate a simpler way of integrating with RoboNetSim. Namely, we show a minimum number of steps that must be taken in order to 'plug in' a robotic simulator into the proposed architecture by only acting at the level of robot controllers. As a practical example, we consider the plug-in of the *Stage* [13, 14] robotic simulator.

In order to show in practice the importance of realistic network simulation and the impact of different communication models, we employ RoboNetSim in the context of simple, yet paradigmatic tasks for multi-robot cooperation and coordination. We design a distributed task-assignment problem and a coordinated mobility problem, and we show that simulating networking issues with different degrees of accuracy results in different behaviors and performance. In turn, these would have a clear impact on the final claims associated to a scientific work in multi-robot/robot-swarm domains.

The rest of the paper is organized as follows. In Section 2 we discuss related work. We focus on the relation between network simulations and reality (Section 2.1), on the scalability of network simulators (Section 2.2), and on the simulation of networked multi-robot systems (Section 2.3). In Section 3 we present RoboNetSim as a general framework for integrating robotic simulators with discrete-event network simulators. Section 4 demonstrates the application of the RoboNetSim architecture in two integrated simulation environments. In Section 4.5 we provide a step-by-step guide on how to simply plug-in a robotic simulator in our framework only acting on robot controllers. In Section 5 we analyze the proposed tools in terms of computational performance, correctness, and generated overhead. Sections 6 and 7 present two case studies showing how different simulated communication models affect the operation of multi-robot teams. Finally, in Section 8 we draw conclusions and outline future work.

2. Related Work

2.1. Reliability of Network Simulations

We start discussing related literature focusing on the relations between network simulations and reality. The key question that needs to be answered first is: can existing network simulation tools describe the communication behavior of a networked multi-robot system with the desired/necessary accuracy?

In the context of ad hoc wireless communication, it is well known that real-world results might significantly differ from the results obtained through simulation. In [15] an extensive survey of real-world implementations of mobile ad-hoc networks is presented, showing the discrepancy between simulation and real-world. Various simplifications that are commonly made in simulations and their impact on predicted outcomes are discussed in the context of key findings from real experiments. Similar discussion, supported by routing experiments in large outdoor areas, can be found in [16].

However, while these works seem to indicate, in general, a gap between simulation and real-world, it is also known that when the appropriate models with correct parameters are chosen, simulation can provide important insights into the asymptotic behavior of large networks [17]. A discussion on the level of detail that should be selected in wireless simulations can be found in [18]. In [19], the authors combine a real testbed consisting of driving cars and a proposed emulation technique. They show that NS-2 can accurately simulate network traffic in an ad-hoc network consisting of 16 moving vehicles. An indoor static wireless network simulated in NS-2 is analyzed in [20] showing that packet delivery ratios and the connectivity graphs can be modeled with a high accuracy, providing that the shadowing radio propagation model is used and properly calibrated. In [21] NS-2 simulations related to QoS issues in ad-hoc networks are compared with real data, showing that experimental results are consistent with simulations in terms of overall trends, although simulation results tend to be more optimistic in terms of both delay and throughput measurements.

The accuracy of NS-3 to simulate Wi-Fi networks has been assessed in a number of works. The real-world experiments performed in [22] show that the results of the frame error rate model for OFDM signals in the NS-3 are close to the experimental data. In [23] a detailed validation of the IEEE 802.11 MAC model in NS-3 is provided: while in several cases there are noticeable quantitative differences, in general there is a good qualitative agreement between the simulator and data from the considered testbed.

The accuracy of another network simulator, *SWAN*, has been validated in [24] and [25]: with an appropriate choice of models, the simulator can accurately predict outdoor performance.

To sum up our discussion here, we can say that modern network simulation techniques can be sufficient to give an overall overview on how a typical networked multi-robot system will work in the real world. Although, it is necessary that appropriate models are chosen and that they are properly calibrated, meaning that roboticists might need to build up a professional profile regarding networking issues.

2.2. Scalability of Network Simulations

The most scalable robotic simulators are currently capable of simulating tens of thousands of robots. For instance, Stage can simulate populations of around 100 complex robots or 1,000 simple robots in real time or less, and there are results reported for simulations with 100,000 simple robots [13]. ARGoS can simulate 10,000 simple robots faster than real time [11]. Moreover, in their future work, the authors of ARGoS plan to further improve the scalability to reach real-time performance for swarms composed of hundreds of thousands of robots. Given these numbers, the aim of this section is to answer the following question: how network simulators deal with such a large number of nodes? We focus our study on detailed (packet-level) network simulators.

Most of the commonly used wireless networks simulators are capable of simulating networks of up to hundreds, and some of them up to thousands of nodes in a reasonable time. NS-2 deals efficiently with networks of up to 200 nodes, and then problems arise both with computation time and memory consumption [26]. NS-3 scales better, and can simulate networks with hundreds of nodes in a reasonable time on modern PCs. It also consumes less memory compared to NS-2, and can be used to simulate large networks, but the expenses of quite large simulation times. In the *iTETRIS* project [27], vehicular networks up to 20,000 nodes were simulated in NS-3. However, the simulation of only 40 seconds of network's operations was running for several days. Other simulators (*QualNet* [28], *OPNET* [29], *OMNeT++* [30]) show similar limitations [31].

Some specialized tools exist for simulating large-scale wireless sensor networks. The authors of [32] present a brief review of tools for the simulation of sensor networks consisting of up to 10,000 nodes. They also propose their own simulation tool (*GTSNetS*), which scales up to networks of 200,000 nodes on a single core workstation. However, all these tools are not actually complete network simulators. They focus on modeling certain important features of sensor networks, such as sensors and their accuracy, realistic energy models, and specific communication protocols. Typically, they ignore mobility and actuators. Therefore, their application in the field of mobile robotics is limited. Nevertheless, with some extensions, they could be potentially used to simulate large-scale networks consisting of simple robots, or they could be used to simulate certain networking aspects. It is important to point out that some communication issues may be common to both mobile ad-hoc and wireless sensor networks (e.g., radio propagation models and the operation of MAC protocols), while other may significantly differ (e.g., routing algorithms).

A very promising way to increase the scalability of network simulation is the use of parallel and/or distributed simulation tools. *GLoMoSim* [33] is a library-based sequential and parallel simulator for wireless networks. It models all layers of the network protocol stack, and has been used to simulate networks up to 10,000 nodes. Another interesting parallel tool is *SWAN* [34]. SWAN can simulate networks of similar size as GloMoSim, but provided a limited number of communication models. *WiFra* is an example of a fully distributed framework for the detailed simulation of very large-scale wireless networks. The authors of WiFra demonstrate results for 1,000,000 simulated nodes. However, the network is simulated only up to the MAC layer. In [35], the authors explore the potential of graphics processing units (GPUs) in supporting network simulations. Their tool is able to perform simplified simulations on 60,000 nodes (considering only mobility and connectivity under unit disk graph) 25 times faster in the GPU mode than in the CPU mode.

It is also worth to mention here that a distributed wireless module for NS-3 is being developed. A distributed scheduler was already implemented, but it does not support yet full wireless simulations. Namely, a wireless network needs to be simulated on a single machine, and it can be connected with other networks simulated on different machines via point-to-point links (currently, only point-to-points links are allowed between sub-networks simulated on different machines). According to the latest experiments with fixed (wired) networks performed by the US Army Research Laboratory [36], the NS-3 simulator has shown significant promise as a scalable, high-performance and high-fidelity network simulator. In the largest of the performed simulations, a total of 176 computing nodes were able to simulate 360,448,000 network nodes. A distributed NS-3 version supporting fully wireless simulations is supposed to be released in the near future.

Finally, an interesting alternative to native parallel/distributed network simulators is represented by an approach based on interconnecting existing simulators. Such *federated simulations* may include multiple copies of the same simulator (modeling different portions of the network), or entirely different simulators. In [31], the authors discuss available federation tools and demonstrate results of massive simulations, including the simulation of 4 million nodes performed with federated NS-2 simulators on 1536 processors. Although the applicability of the presented tools is for now limited to static topologies, we see its potential also in the application to mobile networking.

2.3. Simulation of Networked Multi-Robot Systems

A relatively large amount of work addresses the simulation of multi-robot systems, and an even larger body of work considers network simulation. However, the number of works that combine these two domains is limited. In particular, none of the existing multi-robot simulators offers advanced communication models, and none of the existing network simulators flexibly supports the simulation of autonomous mobile robotic agents.

We analyzed the wireless communication models available in four popular robotic simulators: Stage, ARGoS, Webots, and USARSim. The current version of the *Stage* [13] simulator does not offer any official wireless communication model. It exists a patch, however, that provides a simple Wi-Fi model for older versions of the simulator (e.g., for Stage 3.2.2), which is however not supported anymore. The model offers five radio propagation models, including a *Simple Model* (based on a predefined maximum communication range), the *Friis Model*, and the *Log Distance Path Loss Model*. *ARGoS* [11] provides interfaces for wireless sensors and actuators, yet no implementations are available in the release version. The development version (internally available for authors) contains an implementation of wireless devices that can simulate a simple communication model based on a predefined range. The commercial *Webots* simulator [37] provides *Emitter* and *Receiver* nodes that can be used to model radio or infra-red emitters and receivers. One can configure a maximum transmission range and, for the infrared emitters, an opening angle of the emission cone. *USARSim* [38], a high-fidelity simulator of robots and environments based on the *Unreal Engine* game engine, also offers some basic simulation of communication. Namely, it is possible to exchange messages between two robots using a *Wireless Communication Server*. The server acts as a middle man for messages passed between the robots, dropping messages and connections between robots when they are not within the communication range. The range is determined using an indoor radio propagation loss model based on the *Wall Attenuation Factor* (a free space path loss model that accounts for the attenuation caused by walls [39]).

A general observation is that all the considered robotic simulators indeed provide only limited tools for simulating communication. The offered accuracy is modest and does not go beyond the physical layer of the OSI model of the network: all these communication models ignore transmission collisions, media access control mechanisms, routing mechanisms, etc..

As a consequence of the above facts, the majority of simulation-based works in multi-robot systems make idealized assumptions about communication. For instance, in [7] the authors assume that messages sent by robots via broadcasting are immediately available to all other robots. In [40], the influence of limited communication range is investigated. However, the approach is still far from realistic communication models that would consider the operations of different network protocols, radio interferences, etc. Similar simplifications are assumed in other works. In [41], the behavior of a distributed coordination algorithm is simulated assuming that robots within each other range can communicate directly, and robots within a connected sub-network can use an idealized multi-hop communication. In [42], [43] and [44], *communication graphs* are used, where two nodes are connected if the distance between them is lower than the assumed communication range.

There exist a few works, described below, that also try to combine both network and robotic simulation tools in order to perform more realistic simulations.

In [45], *PiccSIM* [46], a joint network and control simulation tool combining MATLAB and Simulink with NS-2, is extended to support mobility and applied to realistic simulations of a mobile robot squad. The research focused on applying PiccSIM to compare a single-path routing protocol with a multi-path routing protocol in one specific scenario.

Modelica [47] is similar to PiccSIM, it combines NS-2 with a modeling language for large-scale physical systems. However, this tool has not been applied to robotic scenarios.

An integrated tool consisting of the *Arena* robotic simulator (the precursor of the Player/Stage [13]) and NS-2 was presented in [48]. The system is applied to compare two communication strategies for a group of six robots performing a resource transportation task. The interface between both simulators is similar to ours. However, compared to our work, it was neither generalized to support other simulators nor an analysis of the computational performance was performed. Moreover, simulations were supposed to run no faster than the real-time, while we report about the simulations where up to 200 nodes were simulated faster than the real-time (see Section 5.1).

In [49], the ARMS multi-robot simulator and a network simulator for the DSDV ad hoc routing protocol, both developed by the same authors of the paper, are combined to study a multi-robot rescue system forming a network chain used to relay information to a base station.

MiNT-m [50] is a testbed platform devised to support experimentation for mobile multi-hop wireless networks. It enables reconfiguration on an experiment-by-experiment basis, by putting each testbed node on a mobile robot centrally controlled through a wireless connection. Centralized robot navigation is based on vision and a trajectory planner for collision avoidance.

A relatively new research area related to mobile multi-robot system is that of *vehicular networks*. Despite many obvious differences, the simulation of vehicular networks faces issues similar to those faced in the simulation of networked robots. In particular, there exists a strong need for combining realistic vehicle traffic simulation models with accurate communication models [51]. A number of integrated tools have been therefore proposed to realize such a combination, considering popular network simulators such as NS-2, NS-3, and OMNeT++ [52, 27, 53].

Although all the different approaches that we have discussed contribute with ad hoc solutions to specific problems, to the best of our knowledge, a general framework for the integration of multi-robot and network simulation does not exist. Moreover, no dedicated tools for multi-robot systems have proved to realistically simulate networked scenarios with hundreds of robots, as in the case of robot swarms. In the large majority of the previous studies, the adopted tools were not evaluated in terms of the computational performance, correctness, and overhead in comparison to simplified communication models. Finally, no direct comparisons were performed in terms of resulting behaviors between advanced communication models and simplified ones in the context of multi-robot systems. In this paper, we attempt to cover all of these issues.

3. RoboNetSim: An Integrated Simulation Framework

In this section we describe RoboNetSim, our general framework that makes it possible to simulate networked multi-robot systems in a more realistic way, emphasizing the communication aspects. As both network simulation and multi-robot simulation are well established domains of study, the RoboNetSim framework assumes the application of state-of-the-art simulation tools. Namely, we simulate the robotic issues (sensors, actuators, physics engines, etc.) in a dedicated multi-robot simulator, and simulate the communication issues (radio propagation models, network protocols, etc.) in a dedicated network simulator.

It is important to point out that the use of such an integrated simulation tool is not always necessary or justified. In some cases, making idealized assumptions about communication can be a reasonable approach. For instance, when only a few robots are being used and they operate in a relatively small and uncluttered room. In such cases, it is enough to use the simplified communication models available in many robotic simulators. Similarly, when considering a mobile robotic network composed of very simple robots operating in a simple, relatively uncluttered environment, it is reasonable (and easier) to implement robots' behaviors and mobility patterns directly inside a network simulator. We claim that the proposed integrated simulation environment should be used when the two following conditions are met simultaneously:

1. there is the need for the accurate modeling of the individual robots (i.e., of their sensors and actuators) and/or of the complex interactions between robots and their environment;
2. there is the need for the accurate modeling of communication, as it is intrinsically not fully reliable, and it directly influences robot behaviors.

Nevertheless, we believe that the large majority of modern distributed robotic systems meet both these conditions, and it is therefore reasonable and necessary to combine robotic and network simulators.

Robotic simulators are usually *discrete-time simulators with a constant time step* [2] (e.g., Stage, ARGoS, Gazebo [54]), whereas network simulators are *discrete-event* simulators (e.g., NS-2, NS-3, QualNet [28]). The former type of simulation assumes that the time step is fixed at the beginning of the simulation, time advances in equal increments, and the state of the system is updated periodically. In the discrete-event simulations the system changes its state in response to events, the simulated time advances from one event to the next, with the time between the events constantly changing. RoboNetSim addresses the problem of combining the operation of a time-driven robotic simulator with an event-driven network simulator.

3.1. Information flow

The interaction between the two simulation environments requires solving two fundamental issues. First, simulation time needs to be synchronized between the simulators. Second, all the required data about the system's state must be efficiently exchanged between the simulators. The general scheme of the information flow between the robotic simulator and the network simulator in the RoboNetSim framework is showed on Figure 1.

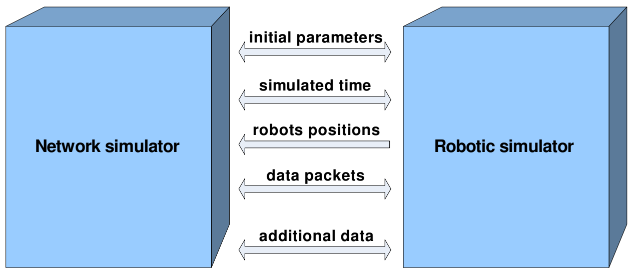


Figure 1: Information flow in the RoboNetSim framework.

In the initial phase, the simulators exchange the parameters of the simulation (e.g., time step, number of robots, characteristics of communication equipment, size of the simulation area) and the network simulator creates a network with mobile nodes corresponding to simulated robots. Then, both simulators periodically exchange the information about their internal simulation time. In addition, at each simulation step, the robotic simulator sends the updated positions of all robots to the network simulator, which in turn updates the positions of the corresponding network nodes. Finally, when robots need to communicate with each other, the corresponding data packets are passed to the network simulator, together with their destinations. The network simulator performs the simulation of the communication, and transfers back to the robotic simulator the data packets that were successfully delivered.

[2]There exist some event-driven simulators also for multi-robot systems. However, they typically model the robots focusing on the functional level. For instance, event-driven simulation can consider the operation of a complex industrial system (e.g., a whole factory), with the robots seen as elements that can produce and service events.

The general scheme above includes the exchange of the minimal, strictly necessary set of data to guarantee the coordination between the simulators. RoboNetSim, however, is capable of transferring any type of useful data between the simulators. One example may be the case in which robots' control algorithms and network control algorithms depend on each other. For instance, a routing algorithm can ask a node controller to move the robot towards a certain direction in order to locally improve network connectivity. Similarly, robotic controllers could use network information about traffic congestion to decide whether to exchange only strictly necessary information or send large amount of data into the network. The information about failures (e.g, at the MAC layer) could also be exchanged and used to take appropriate actions at the controller (e.g., controllers could increase the redundancy of mission critical information which is being sent into the network).

Another notable example of additional data exchange between the simulators and of their mutual dependencies, regards the presence of physical objects in the robotic environment, which can affect communication. In general, physical objects (e.g., obstacles, walls) and robots themselves can attenuate the wireless signal, generate noise and reflections, etc. RoboNetSim is capable of sending information about the presence of such objects to the network simulator, where it can be used to improve the accuracy of the communication model. [3]

3.2. *The Architecture*

In order to realize the above information flow, we designed the following architecture (Figure 2). The schedulers of both simulators are connected with each other and exchange the synchronization data and the information about robots' positions. For each simulated robot, there exists an application at the application layer of the simulated network. The application is installed on the network node corresponding to the robot. There exists a connection between this application and the objects that represent robot's communication devices. The connection is used each time the robot needs to send new data and also each time the application receives data that need to be passed to the robot (i.e., when the network simulator indicates a successful delivery to the robot).

In principle, any of the inter-process communication techniques could be used for data exchanging between the simulators, including sockets, message queues, shared memory, and message passing. In our implementations we decided to use the *socket API*. We based our decision on several factors that imply the flexibility of this technique. First, sockets can be used both locally and remotely, enabling the distributed simulations as well. Second, sockets are platform independent, thus both simulators can actually work under different operating systems. Finally, sockets provide both blocking and unblocking communication interface, making it possible to synchronize the simulated times in both simulators, and provide, at choice, both fully reliable connections (TCP) and potentially unreliable, but with less overhead, connections (UDP).

[3]Unfortunately, the capability to take into account the presence of obstacles is not commonly available in current network simulators. Therefore, we will exploit this direction in future work, since it requires the adaptation of the radio propagation models in the network simulators.

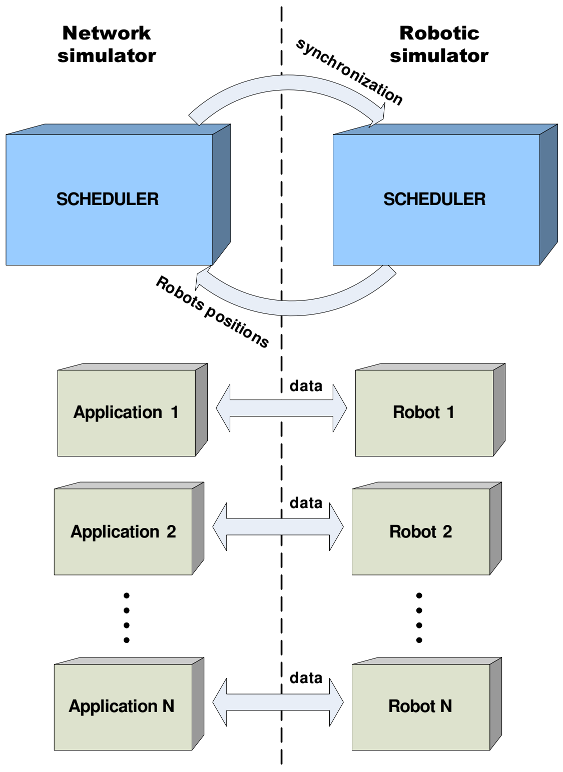


Figure 2: Architecture of the RoboNetSim framework.

While being extremely flexible, the use of sockets also come with certain limitations. In particular, from the one hand, using a large number of sockets might have an impact on the computational performance, and from the other hand, the number of socket ports is constrained by various factors, including the available memory and the encoding of a port variable (which is a 16-bit number in Linux). In principle, when both the network and the multi-robot simulation processes are on the same machine, the performance of inter-process communication could be improved by using shared memory or message queues instead of sockets. However, message queues might present drawbacks similar to those related to the use of sockets when a large numbers of messages is being present in the queue. Therefore, the best way to improve the computational performance and efficiency of our framework would be the use of shared memory, that however would require an adequate and well designed access mechanism (e.g., based on semaphores). On the other hand, our results in Section 5 show that the overall performance of the system is dominated by other factors, namely the internal computation time of both simulators. Also, the limited number of ports (in theory, 65536 ports in Linux) is not an issue in the case of our current implementation, as the simulation of networks with thousands of nodes is currently out of reach for both NS-2 and NS-3. However, taking into account the discussion on the scalability of network simulators (Section 2.2), in the future we plan to extend our architecture in order to support massive multi-robot simulations (swarms composed of hundreds of thousands of robots). In particular, we see the idea of 'aggregating' multiple robots using the same port as an easy and effective way to overcome possible limitations related to the use of a very large number of sockets.

To complete the view of the proposed architecture, we present an illustrative timeline diagram (Figure 3). It can be seen that synchronization takes place at the beginning of each simulation step *i*. During the synchronization time, also the information about robots' positions is exchanged. Then, within the simulation step *i*, the robotic simulator performs all its tasks, including reading from a buffer the messages that were delivered during the previous simulation step, and sending new messages to the network simulator. At the same time, the network simulator waits until all new messages corresponding to the simulation step *i* are received from robots (implemented as a *blocking read* operation), holding the simulated time at the beginning of step *i*. Then, it proceeds with simulating communication until reaching the simulated time corresponding to the beginning of the next simulation step $i + 1$. All messages delivered within step *i* are stored in the buffer, from where they will be read by the robotic simulator during the next step (implemented as a *non-blocking read* operation). It may take longer then one time step for a packet to travel through the network from a source robot to the destination robot (e.g., when a new routing path needs to be established). In such case, the packet is stored internally in the network simulator and delivered within subsequent time steps. It is remarkable that a packet cannot be delivered to the destination's controller at the same time step as it was sent, thus the minimal end-to-end delay seen by robots is equal to a single time step.

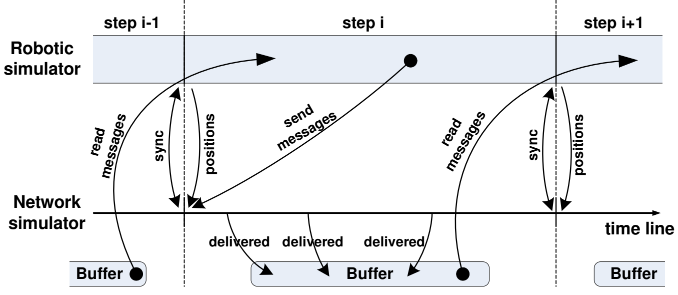


Figure 3: Timeline diagram of the combined simulation.

4. Sample Implementations

In this section, we present two model implementations of RoboNetSim. Both of them fully integrate a network simulator with a robotic simulator by adapting the structure of the two types of simulators. At the user level (the developer of robot controllers) the integration is fully transparent. We also show, in Subsection 4.5, an additional way to perform the integration, which does not require changes to the general structure of the multi-robot simulator, but instead can be realized directly by the user in the code of robot controllers.

4.1. Robot and Network Simulators

In our model implementations, we used three existing, open source tools: *ARGoS* [11, 12] for multi-robot simulation, and *NS-2* [9] and *NS-3* [10] for network simulation. For the integration of a robotic simulator with our framework operating only at controller level, we used the popular Stage [13] open source simulator.

ARGoS is a multi-physics-based simulator for large heterogeneous robotic swarms. It is a highly scalable, parallel, and easily extensible tool programmable in C++. It provides ready-to-use models of several different robots, and the controllers written in the simulation environment can be directly ported to real robots. According to [11, 12] ARGoS can simulate thousands of simple wheeled robots faster than real-time. ARGoS is a discrete-time simulator with a (configurable) constant time step, which is a typical architecture for commonly used robotic simulators that deal with similar level of detail. in this study we have chosen ARGoS precisely because it well represents current state-of-the-art in multi-robot simulation and fully meets our requirements for simulating large multi-robot systems.

Stage can simulate a population of mobile robots, sensors and objects in a two-dimensional environment. Stage provides fairly simple, computationally cheap models of lots of devices. Stage is commonly used together with *Player*, a network server for robot control.

NS-2 is one of the most used simulation environments for computer networks. It covers both wired networks and mobile wireless communication networks. NS-2 is an open-source project including a large set of contributed models for all network layers. NS-2 is based on two programming environments: C++ for computations and OTcl for configuration.

NS-3 is an independent successor of NS-2. Despite the similar name, it has a significantly different design. It is still a discrete-event simulator, yet it was created from scratch and written entirely in C++. It is better designed and more efficient than NS-2, and provides some more accurate communication models (e.g., of 802.11 networks). However, it is a relatively new project and it still offers less ready-to-use models than NS-2 (e.g., at the moment it does not support 802.15 networks).

4.2. ARGoS Interface

From the user's perspective, the core functionality of ARGoS is the possibility of writing controllers for various types of robots. Controllers are C++ programs that are called once per every simulation step. Therefore, in every simulation step, the controller can read the state of sensors, perform its internal logic, and send commands to actuators.

We provide an additional sensor and an additional actuator, which are the only extensions of ARGoS visible to user. Namely, we provide an ***external wireless sensor*** and an ***external wireless actuator*** that together represent a wireless communication device. In the robot controller, users can write messages (together with their destinations) to the new actuator, and read the messages delivered to the robot from the new sensor. The whole integration with a network simulator is fully transparent for users.

When a user decides to equip a robot with the wireless communication device, a new ***wireless entity*** is associated with the robot. The wireless entity is another extension of ARGoS, yet invisible to the user. The entity communicates directly with the network application installed on the network node that corresponds to the given robot (at the network simulator's side). At every simulation step, before the controllers' code is invoked, the wireless entity reads all the messages that were sent from the network application and were addressed to a given robot and makes them accessible for the wireless sensor. Similarly, after the controllers' code is executed, it sends all new messages from the wireless actuators to the network application.

We also provide a ***wireless medium*** as an additional component. A medium in ARGoS is a virtual entity, that does not find a counterpart in the physical space which is simulated, but that can interact with other simulated objects. The wireless medium is responsible for initializing the communication with the network simulator, exchanging the simulation parameters, and then periodically synchronizing the simulated time with the scheduler of the network simulator. It also sends the information about robots' coordinates to the network simulator.

4.3. NS-2 Interface

The NS-2 interface consists of three core elements: an ***ARGoS synchronizer***, an ***ARGoS agent***, and an ***ARGoS generator***. The ARGoS synchronizer is an object responsible for the communication and synchronization with the wireless medium (at the ARGoS simulator side). It also updates mobile nodes positions in the simulated network, according to the information received from the wireless medium.

The ARGoS agent is a network application installed on each node of the simulated network. It communicates directly with the wireless entity of the corresponding robot (at the ARGoS side). At the beginning of every simulation step, the ARGoS agent waits until it receives all new messages generated by the associated robot (or the information that no new messages are available). Only after all agents have received such information, the simulated time in the network simulator proceeds. All new messages are immediately passed to the appropriate ARGoS generator (an NS-2 traffic generator object), which creates corresponding NS-2 packets and sends them out through the network. When the ARGoS agent of the destination node receives the packet, it immediately sends it to the corresponding robot (at the ARGoS side).

4.4. NS-3 Interface

NS-3 has a significantly different architecture compared to NS-2: it is a C++ library providing a set of network simulation models implemented as C++ objects and can be loaded by any user application. Therefore, only two components are required for NS-3 to cooperate with ARGoS: an ***ARGoS scheduler application*** and an ***ARGoS network application***.

The ARGoS scheduler is the main application that defines the communication scenario, sets its parameters, loads network simulation models, and controls the run of the simulation. At the same time, it is responsible for initializing the communication with ARGoS and exchanging the initial simulation parameters. While the simulation is running, the scheduler periodically triggers the synchronization events, receives the robot coordinates from ARGoS, and updates the positions of network nodes accordingly.

The ARGoS network application is installed on every node of the network and plays exactly the same role as the ARGoS generator and the ARGoS agent in the NS-2 interface. As NS-3 uses its internal socket API to send data between nodes, one single network application can do the job and there is no need for additional traffic generating applications.

4.5. Integration at the Controller-Level: Stage Simulator

In the previous sections we described how to perform a full integration of robotic and network simulators in the RoboNetSim framework. The main advantage of this approach is its full transparency for the end-user. Namely, the user can access the communication capabilities of the network simulator by simply interfacing with communication sensors and actuators, using the same interface as in the case of any other sensor/actuator. However, a functionally equivalent result can be also obtained without developing additional modules for the robotic simulator, as we did in the case of ARGoS. In the following, we show how the integration can be also and easily obtained without changing the structure of the robotic simulator: all the required steps are performed in robotic controllers.

We use the Stage simulator as an application example. We extend a sample *FASR* controller provided with the Stage installation (*fasr.cc* source file). We have chosen FASR because of its simplicity. FASR controls robots that move between a nest and a source of items, avoid obstacles, and transport items to the nest. Our extension provides the possibility of exchanging data between the robots, and employs a network simulator in order to simulate the communication issues. [4]

Integration is realized through the following steps:

1. **Open scheduler sockets.**
   In Stage, the main simulation loop is executed by the *World* object. Therefore, this would naturally be the object to act upon to synchronize the simulation time with the scheduler of the network simulator. However, we do not want to modify the World object, that would require modifying the simulator itself (a task that needs knowledge and privileges that might be out of reach for the normal user). Therefore, we perform the synchronization indirectly in a robotic controller, which is periodically called by the World object at each simulation step. Namely, we designate one of the robots (and call it a *master robot*) to be responsible for communication with the network simulator. Inside the controller of this robot, during the controller's initialization phase, we need to open a communication socket to the network simulator. In our implementation, the network simulator's scheduler expects that the

[4] The modified controller's source files, as well as the source files of the other proposed integration tools are available at: `http://www.idsia.ch/~gianni/SwarmRobotics/simulation/`

connection is established using the *port_base* port number, which can be set as a parameter.

2. **Open application sockets.**
During the initialization phase of a robotic controller, it is required that a connection to the corresponding application at the network simulator side is established. This connection is further used to exchange data packets between the simulators. At this aim, the controller of each robot needs to open a communication socket to the network simulator (including the master robot, that should be already connected to the scheduler of the network simulator). We assign a unique *ID* to each robot (an integer value, starting from 0), and every robot opens a socket on the port number ($port\_base + 1 + ID$), where the corresponding network application is waiting for the connection.
3. **Exchange initial information.**
Before starting the main simulation loop, Stage needs to initialize the connection with a network scheduler. It has to send basic information to the network simulator (e.g., the simulation arena size, the number of robots, etc.) and receive the acknowledgment of reception. We perform this during the initialization of the master robot's controller.
4. **Send a 'start' packet from each robot.**
Similarly, every robotic controller needs to initialize the connection with the corresponding network application by sending a predefined *start packet*.
5. **[in a loop:] Synchronize time, send robots positions.**
At the beginning of each simulation step, the *World* object would be required to wait for a *synchronization packet* from the network simulator. After packet reception, it sends position updates for every robot. Again, we use the controller of the master robot here. Inside the controller, we set up an additional callback in the World object (Stage provides an interface for this). This callback is called by the World object at the end of each simulation step.
6. **[in a loop:] Robot controllers send/receive messages.**
During each simulation step, robot controllers can use their application sockets to send new data packets to other robots (using the unique *IDs* as destination addresses) and/or receive messages sent by other robots.

5. Evaluation Study

In this section, we report the results of an experimental study evaluating two of the presented simulation tools, the integrated ARGoS/NS-2 and ARGoS/NS-3 simulators. We analyze the *performance*, in terms of the consumed CPU time (Section 5.1), and the *overhead* of the proposed architecture. Namely, we study how much CPU time is consumed by the more accurate, integrated simulation environment in comparison to the standalone simulators executing the same scenarios (Section 5.2). We also show that the proposed architecture integrating and synchronizing two different simulation tools is *correct*, in the sense that it does not introduce unexpected errors. In particular, the results produced at both sides of the simulation are exactly the same as the results produced on standalone simulators provided with the same input (Section 5.3).

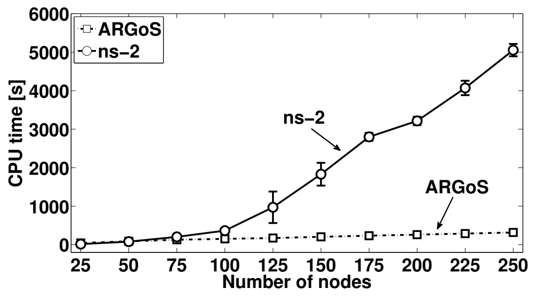
In all studied scenarios, we simulate a group of *foot-bots*, small ground robots developed within the project *Swarmanoid* [55] (`http://www.swarmanoid.org`). ARGoS, developed in the same project, includes accurate models for these robots. Foot-bots move across a squared simulation area at a speed of 1 m/s, performing a simple collision avoidance based on proximity sensors which makes them turn away from each other and from the walls. Robot mobility is determined by a kind of a diffusion process with random initial state (the robots are initially placed in the arena according to a uniform random distribution, and with a random orientation). The simulation arena is limited by fixed walls, and its size depends of the number of simulated robots: we set the basic size to 500×500 m^2 for 50 robots, while for different swarm sizes we adjust the area to keep constant the average node density. In ARGoS, we set the simulation step to 0.1s and we use a 2D dynamics physics engine. In both network simulators, we use their default settings for simulating 802.11 networks. We use AODV [56] as multi-hop routing algorithm, the UDP protocol in the transport layer, and and we adopt traffic patterns that are commonly used in MANET research.

5.1. Computational Performance

The computational performance is evaluated using the following scenario. We vary the number of simulated robots between 25 and 250 (400 in NS-3). Robots form a multi-hop ad-hoc communication network. We randomly choose 20% of nodes to generate a CBR (*Constant Bit Rate*) traffic. Each node sends 5 packets per second to one randomly chosen destination node. The simulation is executed for 1000 seconds. We measure the CPU time consumed by both processes: the robotic simulator process (ARGoS, in single-threaded mode) and the network simulator process (NS-2 or NS-3, both single-threaded). Each measurement is averaged over 10 runs. The measurements are performed on a dedicated PC equipped with the Intel Core i7-2600 CPU (3.40GHz, 4 cores) and 8GB of RAM memory. The results are presented in Figure 4.

It can be seen that the environment using NS-3 achieves a better scalability, which is in agreement with the results presented in the literature [26]. NS-2 is able to simulate scenarios up to 250 robots in a reasonable time, whereas NS-3 effectively deals with scenarios up to 400 robots. Real-time or faster simulation is possible up to 100 nodes (with NS-2) and up to 200 nodes (with NS-3). It is worth noticing that ARGoS is the 'bottleneck' only below a certain number of nodes (75 for NS-2, 100 for NS-3). After that most of the CPU time is consumed by the network simulators. Thus, the possibility of parallel network simulations announced by the NS-3 developers would be desirable and could significantly increase the scalability of the proposed solution.

Overall, the experiments have confirmed that the RoboNetSim framework can offer realistic simulations of networked multi-robot systems consisting of hundreds of robots, that can be performed in a reasonable time on typical PC machines.



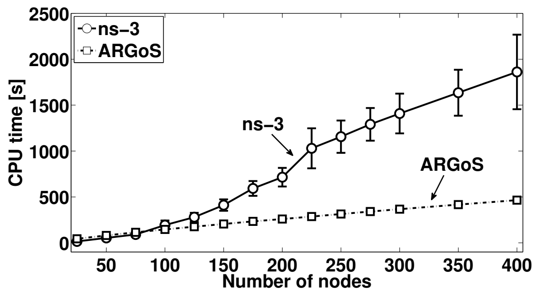


Figure 4: Computational performance in terms of the consumed CPU time. The integrated ARGoS/NS-2 (top) and ARGoS/NS-3 (bottom) environments.

5.2. *Overhead*

Our study of the overhead aims to answer the following question: how much do we need to pay, in terms of the computation time, for more realistic communication models in robotic simulations? We define the overhead as the difference between the total CPU time consumed by the combined network and robot simulators, and the CPU time consumed by the standalone ARGoS with the simplified communication model. We considered the following simulation setups:

- integrated ARGoS/NS-2,
- integrated ARGoS/NS-3,
- standalone NS-2,
- standalone NS-3,
- standalone ARGoS (with a simplified one-hop communication based on a predefined communication range).

We varied the number of nodes between 25 and 200, and we set the simulation time to 200 seconds. The rest of the parameters remained the same as in Section 5.1. Again, we measured the CPU time. The results related to NS-2 are presented in Table 1, and the results related to NS-3 in Table 2.

From the results we can observe that the overhead increases with the number of nodes. Over 100 nodes the overhead is mostly related to the computational requirements of the network simulators. In general, NS-3 generates a lower overhead (roughly half, for the scenarios with more than 100 nodes) compared to NS-2.

As benefits usually come at a cost, the increased accuracy in the simulation of communication requires additional computational effort. However, cost seems reasonable and affordable: for a realistic simulation of 100 networked robots, we 'only' need to pay twice as much the cost for a very simplified one.

Table 1: Overhead of the integrated ARGoS/NS-2 environment, as compared to the standalone simulators (CPU times measured in seconds).

Nodes	ARGoS with NS-2			Standalone	
number	ARGoS	NS-2	sum	ARGoS	NS-2
25	9.1	3.1	12.2	9.5	2.8
50	17.9	16.6	34.5	18.3	16.0
75	26.1	36.3	62.4	27.8	38.5
100	30.2	76.2	106.4	40.5	81.1
125	35.2	217.0	252.2	53.6	212.8
150	41.3	284.4	325.7	68.7	329.6
175	46.3	365.2	411.5	83.5	403.3
200	51.9	393.7	445.6	103.3	567.9

Table 2: Overhead of the integrated ARGoS/NS-3 environment, as compared to the standalone simulators (CPU times measured in seconds).

Nodes	ARGoS with NS-3			Standalone	
number	ARGoS	NS-3	sum	ARGoS	NS-3
25	9.0	2.9	11.9	9.5	3.4
50	17.6	15.2	32.8	18.3	19.2
75	24.8	34.9	59.7	27.8	28.6
100	30.8	52.8	83.6	40.5	53.2
125	37.0	106.7	143.7	53.6	98.7
150	42.2	124.8	167.0	68.7	133.9
175	46.7	150.3	197.0	83.5	174.9
200	52.7	259.5	312.2	103.3	288.5

5.3. *Correctness*

To validate the correctness of the proposed tools, we performed the following study. We selected a number of different simulation scenarios and we executed them in both the integrated simulation tools, and in the standalone NS-2/NS-3 simulators with exactly the same mobility and traffic generation patterns. To achieve this, during the simulations with integrated simulators we gathered detailed mobility traces and we reused them in the standalone NS-2/NS-3 simulators. We forced traffic generators to create the same data packets as were generated by robots, and to create them at the same simulated time. Finally, we carefully set the same seeds for the random number generators.

The resulting communication traces generated by standalone NS-2/NS-3 simulators were directly compared with the ones generated by the combined ARGoS and NS-2/NS-3 environments. We also compared the output of the simulations, in terms of observed delays and packet delivery ratios. The results showed that precisely the same values were obtained for both standalone and integrated simulators (as such, we omit the plots). Therefore, this confirms the correctness of the proposed integrated tools.

However, there is one important remark to be made here. We analyzed the correctness from the perspective of network simulators. Yet, one needs to remember that from the robotic simulator's point of view there is some inherent error introduced by the fixed value of the simulation step. There are two main effects that can be observed: effect of delays in the robotic simulator and effects related to imprecise mobility information passed to the network simulator.

The first effect can be illustrated as follows. The robotic simulator cannot generate packets with a higher frequency than the assigned simulation step. At the same time, the delivered data packets are always received with some delay depending on the time duration of the simulation step. Assuming that in the network simulator packets reach their destinations at times that are randomly and uniformly distributed over time, than the average delay caused by this effect should be equal to the half of the duration of the simulation step. Simulation results for the considered mobility and traffic patterns confirm this (Table 3).

Table 3: Average delay between receiving a packet in the network simulator and receiving the same packet in the robotic simulator, measured for different values of the simulation step (integrated ARGoS/NS-2 environment).

Simulation step [sec]	Averaged delay [sec]
0.01	0.005
0.05	0.026
0.1	0.052
0.2	0.14
0.5	0.25
1.0	0.54

The second effect is connected with the frequency of position updates, determined by the simulation step. Even small differences in mobility can influence connectivity in the network at the given time. In turn, this can influence all simulation results. To study this effect, we repeated the same experiment several times, varying the frequency of position updates at the network simulator's side. The results in terms of averaged packet delays were similar in all experiments, yet not exactly the same. In Figure 5 we present results for three selected simulation step sizes: 0.01, 0.02 and 1.0 seconds.

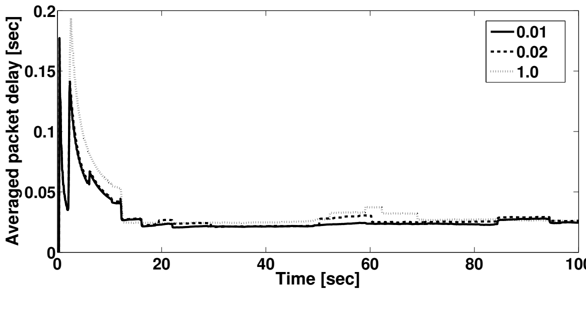


Figure 5: Packet end-to-end delay, averaged in a moving window. The same experiment repeated for three values of the simulation time step: 0.01, 0.02 and 1.0 seconds (integrated ARGoS/NS-2 environment).

Therefore, in general, the value of the simulation step need to be carefully chosen, taking into account the dynamics of robots mobility, the dynamics of communication (and its influence on the system), and the available computational resources. We study the influence of the simulation step on the operation of the considered multi-robot system in Section 6.2.

6. Application Scenario 1: Coordinated Mobility

In this section, we use one of the proposed tools to show that the selected communication model indeed affects the operation of a multi-robot system in a significant way, determining a bias in the final results and the way they are claimed out. More specifically, we use the integrated ARGoS/NS-2 environment in a coordinated mobility scenario. We design a task aimed to emphasize the impact of one-to-one unicast communication between two specific robots. We also enforce the use of multi-hop communication, in order to make the task more difficult and make network simulation play a major role. We compare the results of the NS-2 communication model with simplified communication models, and we highlight the resulting difference both in performance and behavior. We also study how the simulation step size affects the operation of the considered system, and we provide some practical recommendations on how to set this parameter.

6.1. The Problem and the Coordination Scheme

Two selected robots need to coordinate their actions remotely. One of them is termed *master* and defines the mobility pattern. The other is termed *slave* and is supposed to imitate the same mobility pattern of the master, but on a different area. For simplicity, we assume that robots are aware of their own position and orientation. The master periodically sends the information about its current destination to the slave (once per second). The master and the slave are out of their communication range. Hence, they need to use the other robots in the system to relay their messages. The other robots perform their own tasks and also communicate with each other. This is simulated by letting them moving according to random mobility patterns (based on the Random Waypoint mobility model) and generating data according to random traffic generation patterns (a subset of the robots generate CBR traffic to random destination robots).

As a result, a mobile ad-hoc network is formed among the robots. The performance of the system is evaluated in terms of the absolute error between the desired and the actual position of the slave under the following three communication models:

- *Ideal Model:* all robots are within 1-hop communication range and there are no communication errors;
- *Simple Model:* a simple disk model with a fixed communication range (set to 120 meters), which does not consider interferences, collisions, etc.;
- *802.11 Model:* the realistic 802.11 Wi-Fi model from NS-2, with a transmission range set to 120 meters.

As in previous experiments, we use AODV [56] as multi-hop routing algorithm and the UDP protocol in the transport layer. In ARGoS, we set the simulation step to 0.1s and we use a 2D dynamics physics engine.

6.2. Experimental Results

We simulated the above scenario for 22 foot-bots (20 robots + master + slave). Robots move within a wall enclosed area of 500×500 m^2, with a maximum speed of 10 m/s. 10 nodes were used as CBR sources, each one generating 50 packets per second (packet size is set to 70 bytes). The simulation was executed for 1000 seconds. Results are averaged over 10 simulation runs. We analyzed the above mentioned averaged absolute error as a function of time, as well as the distribution of the absolute error, visualized as the empirical cumulative distribution function (CDF). The distance between the master and the slave is set to 250 meters. Therefore, a minimum number of 3 hops is required for communication. We also validated the results in a bigger network of 122 robots placed on an area of 1000×1000 m^2 (5 or more hops are required for communication). In this case, 60 CBR traffic sources were active, each one generating 10 packets per second. Results are shown in Figure 6.

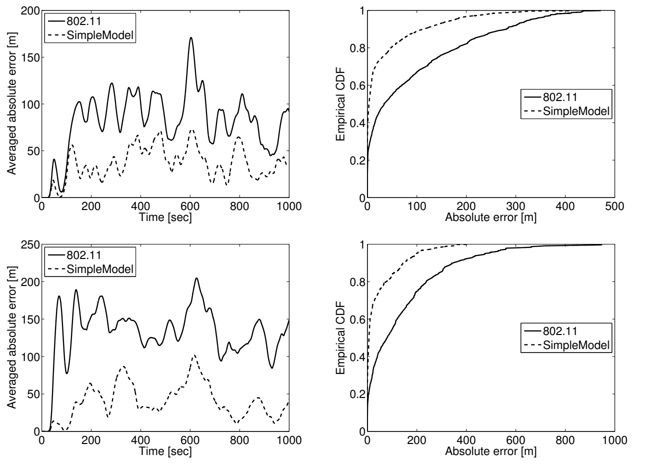


Figure 6: Performance in terms of the absolute error between the desired and the actual position of the slave. The error as a function of time (left), and the error distribution as an empirical CDF (right). Results for 22 robots (top) and for 122 robots (bottom).

In both scenarios, the results speak for themselves. On the plots, we skipped the results of the Ideal Model: when we assume perfect communication, the slave imitates the master almost perfectly and the absolute error is always within a few meters. When we introduce the limited communication range and multi-hop routing, performance decreases significantly (the Simple Model). Going further and introducing the realistic model including radio interferences, collisions, and the operation of the 802.11 MAC protocol, results further deteriorate (the 802.11 Model). It is therefore apparent that the level of detail in the communication model significantly affects the operation of a networked multi-robot system.

To study the effect of the *simulation step size* (set in the robotic simulator side) on the final results, we consider the same scenario above with 22 robots, but decreasing the maximum robot speed from 10 m/s to 5 m/s, as the former value could be too high for simulations with step size greater than 0.1 s (moving at high speed, the robots would travel a relatively large distance at each time step, such as their trajectory would be equivalent as making "jumps"). We repeated the same experiment several times, with different simulation steps, forcing the robots to always follow the same Random Waypoint mobility pattern: in each experiment run the robots visit the same waypoint locations. Results for an individual run are shown in Figure 7. In principle, the lower the step size, the more accurate is the simulation from the robotic side, and the more accurate is the relationship between what is simulated in the robot simulator and what is simulated in the network simulator.

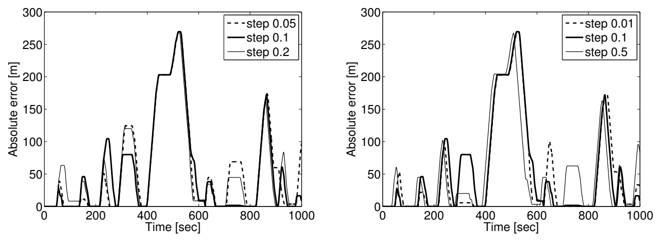


Figure 7: Comparison of performance using different simulation step sizes: 0.05, 0.1, 0,2 (left) and 0.01, 0.1, 0.5 (right). Results for 22 robots and the 802.11 Model. The scenario is the same as in Figure 6, but with a maximum robot speed of 5 m/s. Results are shown in two separate plots for better readability.

We observed that, although the results for the different step sizes are qualitatively similar to each other, some discrepancies in performance can be evidenced, especially when comparing the results for small and relatively large step sizes. As mentioned in Section 5.3, two are the main effects of changing the time step: an additional time delay in delivering transmitted packets from the network to the robotic simulator, and imprecision in the information related to robot mobility (in the network simulator node positions get updated with a delay corresponding to the time step). The results in Figure 7 seem to indicate the dominance of the latter effect on performance. In fact, if the delay shift in delivering transmitted packets would have a major effect on performance, we would observe a systematic shift among the time-dependent curves for the absolute error, with the shift being proportional to the difference among the step sizes. This systematic shift is not evidenced in the figure, only some relative shift of minor entity can be observed. On the other hand, the effect of a misalignment between robot and node positions results in robot and node networks with (slightly) different local topology. In wireless networks, a little difference in topology can easily result in the disruption or in the new availability of communication paths, producing a variation in performance which is more or less random, or, more in general, hard to predict. This is precisely what we observe in Figure 7: the use of different step sizes produces non-systematic variations in performance, without a clear dependence on the magnitude of the used step size (e.g., between 700 and 800 seconds, robot positions are such that the network is intermittently connected, and the differences in step size result in quite different performance, with no clear correlation between step size and performance, precisely because a small variation in position can cause either the availability or the unavailability of a multi-hop path to transmit data from master to slave).

Nevertheless, in spite of the differences on performance and behavior determined by the use of different step sizes in the single run (as that shown in Figure 7), when averaged over multiple runs results (not reported here) show that the differences are quite negligible, due to the mostly random nature of the effect of changing step size (in relation to the discrepancies in robot/node positions).

In the last experiment, we compared the results obtained when using two different *routing protocols*: *AODV* and *AntHocNet* [57] (Figure 8). AntHocNet is an adaptive routing algorithm for mobile ad hoc networks derived from the Ant Colony Optimization (ACO) framework. In the experimental evaluation reported in [57] it has shown to outperform AODV in highly dynamic scenarios, such as the one considered here. The results show that AntHocNet indeed reduces the influence of unreliable communication and improves the performance of the system. From this we can draw two important conclusions. First, the choice of a specific network mechanism can significantly affect the operation of a multi-robot system, and need to be carefully taken into account when drawing conclusions. Second, the proposed simulation tools make it possible to effectively analyze the efficacy of various network control algorithms before implementing them on real robots.

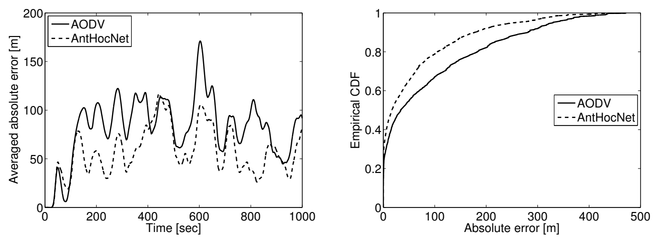


Figure 8: Comparison of performance using different routing algorithms: AODV and AntHocNet. Results for 22 robots and the 802.11 Model. The scenario is the same as in Figure 6.

7. Application Scenario 2: Distributed Task Assignment

The aim of this section is to validate the results presented in Section 6 in a significantly different communication scenario. Namely, in the previous experiments we focused on the one-to-one unicast communication between two specific robots. Here, robots use a flooding mechanism to propagate messages through the network, and all the messages are sent as broadcasts, without a specified destination. This implies that the communication scheme is expected to be quite robust and the influence of communication failures on the operation of a multi-robot team may be limited. We employ the integrated ARGoS/NS-3 environment to perform the experiments.

7.1. The Problem and the Coordination Scheme

We consider the following problem:

- within a wall enclosed area of 500×500 m^2 there are randomly placed events that robots need to service;
- each event requires n robots to be handled, and it takes t seconds for the robots to service the event (we set $n = 4$ and $t = 50$ s);
- robots are initially randomly placed near to one of the walls, they do not know the locations of the events, and they are supposed to explore the area and service all the events as fast as possible;
- robots are able to detect the events within a certain range ($r_{discover} = 50$ m) from their location;
- there are n_{types} types of events that require different skills to be handled (and there are also n_{types} of skills). Each robot is initially "equipped" with n_{skills} randomly chosen skills (we set $n_{types} = 2$ and $n_{skills} = 1$);
- in addition, and to penalize "pointless" moves, robots have initial energy for operating for t_{energy} seconds ($t_{energy} = 200$). After the energy has fallen down below a certain level, the robot needs to immediately go to a charging place and recharge its batteries. The charging place is located in the center of the arena;
- robots are rewarded for handling the events by recharging their batteries;
- robots may communicate with each other, and their task is to decide whether to explore for new events to be serviced or to assist other robots in handling the already discovered events, reaching the required number.

We use foot-bots with a similar configuration as in Section 5, increasing their maximum speed to 5 m/s in order to simulate a more dynamic scenario. Again, we assume that robots are aware of their own position and orientation. We use the same ARGoS configuration as in the previous sections (simulation step set to 0.1s, 2D dynamics).

We implement a distributed coordination mechanism that is based on exchanging messages between robots. There are five types of messages that a robot can broadcast. A status message S is sent to inform other robots about new discovered events to be handled and also to inform how many robots are currently waiting by the event. A "going to" message G is sent when a robot decides to go and handle the event. A "canceled going" message C is send when a robot changes its decision and gives up going to the event. A message H is sent when enough robots approached to the event and started to service it. An "abandoning" message A is sent when a robot waiting for help decides to abandon the event. All messages contain the following information: the coordinates of the event, the number of robots required to handle the event, the required skill, the number of robots already waiting by the event, and the coordinates of the sender. Robots store the above information in their local memory, and in addition they also store the estimated number of robots that already decided to go to the given event (based on the received G and C messages).

Each robot performs the following operations:

- if there are no events stored in the memory, the robot performs exploration and searches for new events. The search strategy proposed in [7] is implemented, where robots perform a kind of probabilistic exploration with a memory of already explored areas;

- when an event is discovered, the robot broadcasts the S message to inform the others. If the robot has the required skill, it waits for help to handle the event. Otherwise, it continues the exploration;

- other robots that receive the S message update their memory and decide whether to go to the event or not. The decision is based on the information stored in the local memory. Namely, with the highest priority robots choose the closest event between the events that require more robots to be serviced (i.e., the number of required robots is higher then the number of the robots already waiting by the event plus the number of the robots that previously decided to go there). If there are no such events, a robot chooses another event with a certain probability or switch to exploration;

- the above decision is made every time a robot receives a new message and updates its local memory.

7.2. Experimental Results

We simulated the above scenario for 20, 30, and 40 robots. We set the number of events to 5, 10, 15, and 20. Each measurement is an average of 25 runs, and we use confidence intervals and/or the Student's t-Test to validate the statistical significance of the results. We analyzed the performance resulting from different communication models, using *flooding* as baseline performance. The flooding algorithm that we use is based on multi-hop packet relaying and the use of sequence numbers to detect (and remove) duplicate packets, with the effect of minimizing network overloading by limiting the number of circulating packets. The following three communication models were considered:

- idealized one-hop communication, with unlimited transmission range among all robots (referred to as ***all***);

- probabilistic disc model: within a given range (r = 120 m), the transmission is successful with a probability p. We use three values of p, namely $p = 0.5$, $p = 0.75$ and $p = 1.0$ (referred to as ***p50***, ***p75*** and ***p100***, respectively);

- the full 802.11a Wi-Fi model from NS-3 (referred to as ***NS-3***), including both physical channel and the MAC protocol simulation.

We analyze the average task completion time (Figure 9). The first observation is that the applied communication model indeed affects the performance of the considered robotic system, confirming the results from Section 6. For 20 robots, the average task completion time for the realistic model (*NS-3*) can be more than 50% longer than in the case of the ideal communication model (*all*).

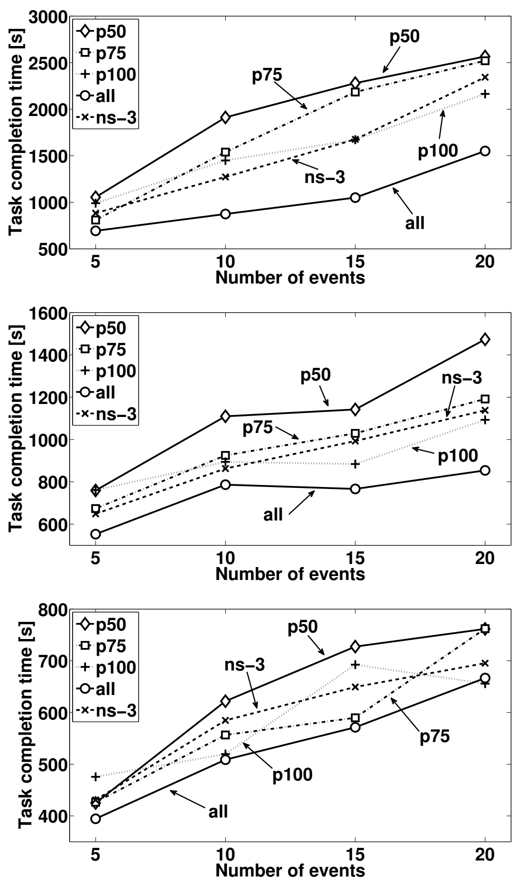


Figure 9: Average task completion time. Results for 20, 30, and 40 robots (top, middle, and bottom, respectively). Error bars are skipped for clarity (see Section 7.2 for details about statistical significance of the results).

It can be also observed that the results depend on the scenario parameters. When we simulate 40 robots, the task is solved relatively fast, regardless of the communication model used. In this case, it is likely that the high density of nodes implied the high connectivity in the network. At the same time, the network was not overloaded as the amount of transferred data was moderate and we used sequence numbers to limit the flooding. As a result, a high level of the delivery ratio was achieved and there is no statistically significant difference in the results obtained for the different models. On the contrary, the differences become visible when simulating 30 nodes: the realistic *NS-3* model obtains similar results as *p75* and *p100*, which are significantly better than the *p50* model and significantly worse than the idealized *all* model (confirmed by statistical tests with p-values less than 0.1). For 20 robots, the results of the realistic model are closer to *p100*. Therefore, the communication seems to be more reliable in this case, which may be explained by the decreased interference implied by the lower density of nodes in the network.

Summarizing the results presented in this section, we can notice that there is a significant influence of the communication model used in simulations on the performance and behavior of the considered multi-robot system. Moreover, it appears that it is not a trivial task to choose a simplified communication model (e.g., a probabilistic disk model) and set its parameters in a way that it would well imitate a more advanced model offered by network simulation tools. In particular, a simplified model that offers accurate results in one scenario, can be inadequate in another scenario.

8. Conclusions and Future Work

In this paper, we proposed *RoboNetSim*: a framework for realistic simulation of networked robotic systems. The framework makes it possible to use advanced network simulation tools together with robotic simulators. We claim that the accuracy of modern network simulators is sufficient to give an overall overview on how a typical communicating multi-robot system will work in the real world. We support this claim by presenting various validation tests of network simulators found in the literature. We presented two case studies that demonstrate the value of the proposed framework.

We believe that the proposed simulation tools can be a step toward better designed and more effective real-world multi-robotic systems. Namely, the robotic controllers prepared in simulation and tested in terms of their robustness to communication issues can be directly ported to real robots, with a reasonable chance of meeting design expectations. In this respect, it is important to remark that the controllers created in ARGoS can be directly uploaded to robots. In a similar way, both NS-2 and NS-3 simulators provide interfaces to the *Click Modular Router* [58]. Therefore, one may implement a new routing algorithm in *Click*, simulate its operation in a multi-robot system using the RoboNetSim framework, and then run exactly the same routing code on any Linux-based robot.

As future work, we see three main directions. The first one is to validate the results of the simulation of large networked multi-robot systems in a real-world experimental testbed. We already have a small fleet of robots and their models in ARGoS, yet we plan to extend it and create a large heterogeneous robotic swarm. This direction may also include a further increase of the simulations realism by introducing and validating the dependency between communication models and objects in the simulated space (e.g., obstacles and robots themselves attenuating signal, generating noise, etc.). The second direction is to extend the proposed architecture in order to support massive multi-robot simulations, composed of hundreds of thousands of robots. Here, we plan to use the distributed version of wireless NS-3 simulations (providing that the corresponding module will be released), or alternatively to study the possibility of federating multiple NS-3 simulators. The third possible future direction would be to employ RoboNetSim in order to investigate the operation of various network control algorithms in large multi-robot scenarios. In particular, we are interested in the performance of adaptive routing algorithms. We also plan to extend RoboNetSim with an interface that would flexibly enable studying feedback mechanisms between robotic controllers and network control algorithms (at various network layers).

References

[1] M. Dorigo, E. Sahin, Guest Editorial. Special Issue: Swarm Robotics, Autonomous Robots 17 (2–3) (2004) 111–113.

[2] L. Bayindir, E. Sahin, A review of studies in swarm robotics, Turkish Journal of Electrical Engineering 15 (2).

[3] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, Cooperative Mobile Robotics: Antecedents and Directions, Autonomous Robots 4 (1997) 226–234.

[4] S. Nouyan, A. Campo, M. Dorigo, Path formation in a robot swarm. Self-organized strategies to find your way home, Swarm Intelligence 2 (1) (2008) 1–23.

[5] F. Ducatelle, G. A. Di Caro, C. Pinciroli, F. Mondada, L. Gambardella, Communication assisted navigation in robotic swarms: self-organization and cooperation, in: Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4981–4988, 2011.

[6] C. Tschudin, P. Gunningberg, H. Lundgren, E. Nordström, Lessons from Experimental MANET Research, Elsevier Ad Hoc Networks Journal 3 (2) (2005) 221–233.

[7] D. Zhang, G. Xie, J. Yu, L. Wang, Adaptive task assignment for multiple mobile robots via swarm intelligence approach, Robots and Autonomous Systems 55 (2007) 572–588.

[8] M. Kudelski, M. Cinus, L. Gambardella, G. A. Di Caro, A Framework for Realistic Simulation of Networked Multi-Robot Systems, in: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2012 (to be published), 5018–5025, Note: a preliminary version was disseminated with Technical Report 6-12, IDSIA, Lugano, Switzerland, 2012.

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

[10] ns-3, Discrete-event network simulator for Internet systems, Web: `http://www.nsnam.org/`, .

[11] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, T. Stirling, A. Gutierrez, L. Gambardella, M. Dorigo, ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics, in: Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 5027–5034, 2011.

[12] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, M. Dorigo, ARGoS: A Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems, Swarm Intelligence 6 (4) (2012) 271–295.

[13] B. Gerkey, R. Vaughan, A. Howard, The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, in: Proceedings of the 11th Int. Conf. on Advanced Robotics (ICAR), 317–323, 2003.

[14] R. T. Vaughan, Massively Multi-Robot Simulations in Stage, Swarm Intelligence 2 (2-4) (2008) 189–208.

[15] W. Kiess, M. Mauve, A survey on real-world implementations of mobile ad-hoc networks, Ad Hoc Networks 5 (3) (2007) 324–339.

[16] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, C. Elliott, Experimental evaluation of wireless simulation assumptions, in: Proceedings of the ACM Symp. on Modeling, analysis and simulation of wireless and mobile systems, 78–82, 2004.

[17] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, J. Zahorjan, Measurement-based models of delivery and interference in static wireless networks, in: Proceedings of the ACM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications, 51–62, 2006.

[18] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K.-C. Lan, Y. Xu, W. Ye, D. Estrin, R. Govindan, Effects of Detail in Wireless Network Simulation, in: In Proceedings of the SCS Multiconference on Distributed Simulation, 3–11, 2001.

[19] D. Johnson, Validation of Wireless and Mobile Network Models and Simulation, in: Proc. of the DARPA/NIST Network Simulation Validation Workshop, 1999.

[20] S. Ivanov, A. Herms, G. Lukas, Experimental validation of the ns-2 wireless model using simulation, emulation, and real network, in: In 4th Workshop on Mobile Ad-Hoc Networks (WMAN07), 433–444, 2007.

[21] A. Torres, C. Calafate, J.-C. Cano, P. Manzoni, Deploying a real IEEE 802.11e testbed to validate simulation results, in: Proceedings of the 34th IEEE Conf. on Local Computer Networks, 109–115, 2009.

[22] G. Pei, T. R. Henderson, Validation of OFDM error rate model in ns-3, Tech. Rep., Boeing Research & Technology, 2010.

[23] N. Baldo, M. Requena-Esteso, J. Núñez Martínez, M. Portolès-Comeras, J. Nin-Guerrero, P. Dini, J. Mangues-Bafalluy, Validation of the IEEE

802.11 MAC model in the ns3 simulator using the EXTREME testbed, in: Proceedings of the 3rd Int. ICST Conf. on Simulation Tools and Techniques, 64:1–64:9, 2010.

[24] R. S. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, S. McGrath, Y. Yuan, Outdoor experimental comparison of four ad hoc routing algorithms, in: Proceedings of the 7th ACM Int. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 220–229, 2004.

[25] J. Liu, Y. Yuan, D. M. Nicol, R. S. Gray, C. C. Newport, D. Kotz, L. F. Perrone, Empirical Validation of Wireless Models in Simulations of Ad Hoc Routing Protocols, Simulation 81 (4) (2005) 307–323.

[26] M. Ikeda, E. Kulla, L. Barolli, M. Takizawa, R. Miho, Performance Evaluation of Wireless Mobile Ad-Hoc Network via NS-3 Simulator, in: Proceedings of the 14th International Conference on Network-Based Information Systems (NBiS), 135–141, 2011.

[27] J. Härri, P. Cataldi, D. Krajzewicz, R. J. Blokpoel, Y. Lopez, J. Leguay, C. Bonnet, L. Bieker, Modeling and simulating ITS applications with iTETRIS, in: Proceedings of the 6th ACM workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N), 33–40, 2011.

[28] QualNet Simulator, Scalable Network Technologies, Web: `http://www.scalable-networks.com`, .

[29] Opnet Modeler, OPNET Technologies, Inc., Web: `http://www.opnet.com`, .

[30] A. Varga, R. Hornig, An overview of the OMNeT++ simulation environment, in: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, 60:1–60:10, 2008.

[31] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. H. Ammar, G. F. Riley, Large-Scale Network Simulation: How Big? How Fast?, in: 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 116–123, 2003.

[32] E. M. Ould-Ahmed-Vall, G. F. Riley, B. S. Heck, Large-Scale Sensor Networks Simulation with GTSNetS, Simulation 83 (3) (2007) 273–290.

[33] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, M. Gerla, GloMoSim: A Scalable Network Simulation Environment, UCLA Computer Science Department Technical Report 990027.

[34] J. Liu, F. L. Perrone, D. M. Nicol, M. Liljenstam, C. Elliott, D. Pearson, Simulation Modeling of Large-Scale Ad-hoc Sensor Networks, in: European Simulation Interoperability Workshop, 2001.

[35] B. Romdhanne, B. N. Navid, Cunetsim: a new simulation framework for large scale mobile networks, in: Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques, 217–219, 2012.

[36] K. Renard, C. Peri, J. Clarke, A performance and scalability evaluation of the ns-3 distributed scheduler, in: Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques, 378–382, 2012.

[37] O. Michel, Webots: Professional Mobile Robot Simulation, Journal of Advanced Robotics Systems 1 (1) (2004) 39–42.

[38] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper, USARSim: a robot simulator for research and education, in: Proceedings of the IEEE Int. Conf. on Robotics and Automation, 1400–1405, 2007.

[39] P. Bahl, V. N. Padmanabhan, RADAR: An In-Building RF-Based User Location and Tracking System, in: Proceedings of the IEEE Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM), vol. 2, 775–784, 2000.

[40] A. R. Mosteo, L. Montano, M. G. Lagoudakis, Multi-robot routing under limited communication range, in: Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA), 1531–1536, 2008.

[41] W. Sheng, Q. Yang, J. Tan, N. Xi, Distributed multi-robot coordination in area exploration, Robotics and Autonomous Sys. 54 (12) (2006) 945–955.

[42] M. Ji, M. Egerstedt, Distributed Coordination Control of Multiagent Systems While Preserving Connectedness, IEEE Transactions on Robotics 23 (4) (2007) 693–703.

[43] M. Zavlanos, A. Jadbabaie, G. Pappas, Flocking while preserving network connectivity, in: Proceedings of the 46th IEEE Conf. on Decision and Control, 2919–2924, 2007.

[44] L. Xiaoli, X. Yugeng, Flocking of multi-agent dynamic systems with guaranteed group connectivity, in: Proceedings of the 27th Chinese Control Conference (CCC), 546–551, 2008.

[45] M. Pohjola, S. Nethi, R. Jantti, Wireless control of mobile robot squad with link failure, in: Proceedings of the 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 648–656, 2008.

[46] S. Nethi, M. Pohjola, L. Eriksson, R. Jantti, Platform for Emulating Networked Control Systems in Laboratory Environments, in: Proceedings of the IEEE Int. Symp. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 1–8, 2007.

[47] A. Al-Hammouri, V. Liberatore, H. Al-Omari, Z. Al-Qudah, M. S. Branicky, D. Agrawal, A co-simulation platform for actuator networks, in: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys), 383–384, 2007.

[48] W. Ye, R. Vaughan, G. Sukhatme, J. Heidemann, D. Estrin, M. Matarić, Evaluating control strategies for wireless-networked robots using an integrated robot and network simulation, in: Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2941–2947, 2001.

[49] H. Sugiyama, T. Tsujioka, M. Murata, Autonomous chain network formation by multi-robot rescue system with ad hoc networking, in: Proceedings of the IEEE International Workshop on Safety Security and Rescue Robotics (SSRR), 1–6, 2010.

[50] P. De, A. Raniwala, R. Krishnan, K. Tatavarthi, J. Modi, N. Syed, S. Sharma, T. Chiueh, MiNT-m: an autonomous mobile wireless experimentation platform, in: Proc. of the 4th ACM Int. Conference on Mobile Systems, Applications and Services (MobiSys), 124–137, 2006.

[51] R. Stanica, E. Chaput, A.-L. Beylot, Simulation of vehicular ad-hoc networks: Challenges, review of tools and recommendations, Computer Networks 55 (14) (2011) 3179–3188.

[52] M. Piórkowski, M. Raya, A. L. Lugo, P. Papadimitratos, M. Grossglauser, J.-P. Hubaux, TraNS: realistic joint traffic and network simulator for VANETs, ACM SIGMOBILE Mobile Computing and Communications Review 12 (1) (2008) 31–33.

[53] C. Sommer, Z. Yao, R. German, F. Dressier, Simulating the influence of IVC on road traffic using bidirectionally coupled simulators, in: Proceedings of IEEE INFOCOM Workshops, 1–6, 2008.

[54] N. Koenig, A. Howard, Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator, in: Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2149–2154, 2004.

[55] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. A. Di Caro, & other 22 authors, Swarmanoid: a novel concept for the study of heterogeneous robotic swarms, IEEE Robotics & Automation Magazine, 2012 (to be published) .

[56] C. Perkins, E. Royer, Ad-hoc On-Demand Distance Vector Routing, in: In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, 90–100, 1997.

[57] G. A. Di Caro, F. Ducatelle, L. M. Gambardella, AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks, European Transactions on Telecommunications 16 (2005) 443–455.

[58] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, The click modular router, ACM Trans. on Computer Sys. 18 (3) (2000) 263–297.