Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2010, Article ID 149075, 20 pages
doi:10.1155/2010/149075

Research Article

W-NINE: A Two-Stage Emulation Platform for Mobile and Wireless Systems

Emmanuel Conchon,[1] Tanguy Pérennou,[2, 3] Johan Garcia,[4] and Michel Diaz[2, 3]

[1] *Université de Toulouse, IRIT/IRT-Champollion/ISIS-ISAE, 81100 Castres, France*
[2] *CNRS, LAAS, 7 Avenue du colonel Roche, 31077 Toulouse, France*
[3] *Université de Toulouse, UPS, INSA, INP, ISAE, LAAS, 31077 Toulouse, France*
[4] *Department of Computer Science, University of Karlstad, SE-651 88 Karlstad, Sweden*

Correspondence should be addressed to Emmanuel Conchon, emmanuel.conchon@irit.fr

Received 31 May 2009; Accepted 9 November 2009

Academic Editor: Nikos Passas

More and more applications and protocols are now running on wireless networks. Testing the implementation of such applications and protocols is a real challenge as the position of the mobile terminals and environmental effects strongly affect the overall performance. Network emulation is often perceived as a good trade-off between experiments on operational wireless networks and discrete-event simulations on Opnet or ns-2. However, ensuring repeatability and realism in network emulation while taking into account mobility in a wireless environment is very difficult. This paper proposes a network emulation platform, called W-NINE, based on off-line computations preceding online pattern-based traffic shaping. The underlying concepts of repeatability, dynamicity, accuracy, and realism are defined in the emulation context. Two different simple case studies illustrate the validity of our approach with respect to these concepts.

1. Introduction

Testing and evaluating a transport protocol or a distributed application is a challenging task for researchers. Indeed, when a new protocol/application is proposed, its behavior has to be compared to the behavior of existing protocols/application. The testing stage is also important for fine tuning the different mechanisms involved in new protocols/applications. In networking and in wireless networking in particular, several difficult points have to be considered to make such an evaluation. In this work, we have chosen to consider three main points: accuracy, dynamicity, and repeatability and also a fourth point which can be viewed as a composition of the others: realism.

For this purpose, several solutions can be used. First of them is the use of actual tests, where real applications or communication stacks under test are deployed on the real operational machines. The accuracy of this method is high, but there is a lack of control of environmental parameters. For instance, there is no possibility to control the weather conditions when testing a protocol/application in a satellite environment. Furthermore, the experiment cannot be repeated with identical parameters as many times as needed, which is problematic to tune the parameters of the new proposition under test. Such repeatability is highly useful for providing reliable statistical results or to find an optimal operating conditions.

The second solution which is widely used by researchers is simulation, which provides a fully controllable environment ensuring repeatability. A majoronly need bandwidth delays drawback of simulation is the need for many models: application models, protocol models, and traffic models. While some of these may be available, some others can be hard or even impossible to develop (closed source applications). Obtaining the traffic model can also be very difficult especially for multimedia or distributed applications evaluation. Finally, even if the models used for testing are validated, correct behavior of the final implementation has to be verified. For example, it has been shown by [1] that the TCP stack implementation in FreeBSD 6 does not behave

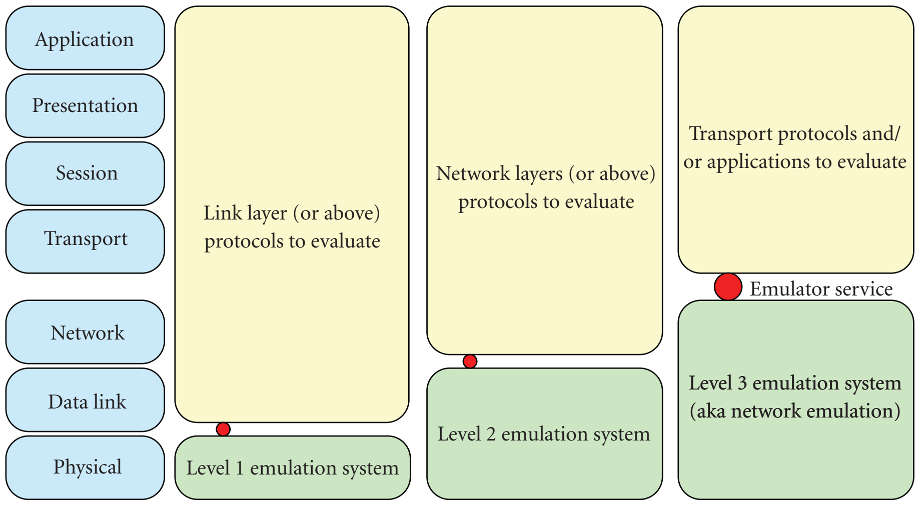

Application
Presentation
Session
Transport
Network
Data link
Physical
Link layer (or above) protocols to evaluate
Level 1 emulation system
Network layers (or above) protocols to evaluate
Level 2 emulation system
Transport protocols and/ or applications to evaluate
Emulator service
Level 3 emulation system (aka network emulation)

Figure 1: Levels of Emulation.

as expected when losses are inserted at specific position in the real traffic, even though the TCP stack model had been previously validated by simulation.

Network emulation is a compromise between real test and simulation that allows the testing of real applications or transport protocols on an adequate experimentation network while in real time mimicking the behavior of another network. A key interest of network emulation (also called IP-level or level 3 emulation in this paper) is that only a few parameters need to be manipulated (typically bandwidth, delays, and losses). The major challenge is to use accurate models to provide accurate emulation parameters while respecting real-time constraints. Some traffic shapers such as Dummynet [2] allow the manipulation of bandwidth, delays, and losses. These traffic shapers do not compute network emulation parameters in real time in order to have a very light performance overhead and manage a large amount of packets. Rather, network emulation parameters are provided manually by the end user or automatically by an external source; in the better case, a traffic shaper can be used as basic tools for an emulation platform. Indeed, it must be noted that most of these traffic shapers do not allow the exact repetition of an experiment, especially for the placement of losses.

In this paper, we propose a solution called W-NINE which deals with accuracy by the addition of a simulation stage before emulation time in order to be able to use accurate models to produce the emulation parameters (bandwidth, delays, and losses). These simulated parameters will then be used during emulation to evaluate the final protocol implementation under test. Based on accurate models, the proposed simulator generates an emulation scenario that synthesizes all the simulated emulation parameters, providing a dynamic behavior of the network to emulate. Moreover, this emulation scenario can be played several times ensuring part of the repeatability. To further enhance repeatability, a traffic shaper supporting precise loss positioning as well as bit-error insertion can be used to reproduce losses and bit-errors. W-NINE emulation relies on a fully centralized architecture that can be viewed as a "black-box" approach.

The paper is organized as follows. Section 2 provides a brief overview of related work. Section 3 discusses the importance of accuracy, dynamicity, repeatability, and realism in network emulation. Section 4 provides an overview of the proposed architecture, and Section 5 illustrates the usefulness of this emulation solution with two different case studies. Finally, Section 7 provides the conclusions.

2. Related Work

Emulation systems largely differ depending on the kind of protocol/application to be tested. For example, the properties of the emulation system must differ between the evaluation of a routing protocol and the evaluation of a streaming application. A routing protocol needs a reproduction of the target network topology while most streaming applications only need information about bandwidth, delays or losses and do not care about the network architecture at the link layer. To deal with this specificity, several levels of emulation, depicted on Figure 1, have been defined.

In Level 1 emulation (Physical Level Emulation), only the physical layer is modified in order to be able to mimic the behavior of the target network. For wireless emulation, most physical-level emulators are based on a wireless infrastructure and use signal attenuators to reproduce signal conditions that can be observed in the wireless network to emulate. With these attenuators, it is then possible to reproduce the target wireless network topology on an experimentation grid. ORBIT [3] is an emulation platform based on this architecture. The main drawbacks of this solution are that the experiment is not fully controllable. Indeed, due to the use of a real wireless infrastructure, Ganu et al. have shown in [4] that the propagation conditions in the platform can be affected reducing the quality of the provided emulation.

To avoid this lack of control caused by interferences, JEmu [5] has proposed the use of a configurable network stack hosting a real implementation of the DSR [6] ad hoc routing protocol and of the 802.11 b MAC layer. JEmu can be seen as a client/server application where the client role is played by a virtual radio layer. This layer is inserted in the configurable protocol stack below the MAC layer and therefore intercepts every packet emitted on the network. These packets are then relayed to a central application that can be viewed as the server. This application then, in real time, simulates node mobility as well as propagation effects to determine if a communication is possible or if collisions might have occurred. If there is no collision, packets are delivered to the destination node; otherwise they can be lost or corrupted according to user configuration. Node mobility can be computed before the experiment so that the amount of computation done in real time is reduced. Nevertheless, the communication support in JEmu is quite simplistic and is only viewed as a communication range. The propagation is limited to the attenuation of the signal based on the distance between the sender and the receiver.

Level 2 emulation (MAC level emulation) is widely used to evaluate real implementations of routing protocols and aims at the simulation in real time of the physical and data link layers. Most of existing solutions are based on a distributed architecture where a virtual MAC layer is embedded on each terminal. This virtual layer needs to provide the same services as a regular data link layer so that the routing protocol under test does not need any modifications. EMWin [7] is a fully distributed wireless emulator focusing on the impact of mobility. For this purpose, an emulated MAC layer has been developed to mimic the medium access CSMA/CA on an ethernet experimentation testbed. Based on neighbor tables, conditions and paths can evolve over time, thus reproducing the dynamic topology of a wireless network. This solution is quite accurate for the mobility purpose, but due to the real-time MAC emulation and the distributed aspect, repeatability is not fully ensured.

Level 3 emulation (Network Emulation or IP-Level Emulation) allows the evaluation of transport protocols or distributed applications. Typically, only three parameters need to be manipulated to mimic specific network conditions: bandwidth, delays, and packet losses. By diminishing the bandwidth, inserting delays, and packet losses, it is possible to reproduce the same conditions as those encountered at the IP level in the network where the experimentation will take place. Traffic shapers such as Dummynet [2] or NISTNet [8] can be used as basic tools to constrain the experimentation traffic according to user specified rules. These rules are expressed in terms of bandwidth restriction, delays, and losses insertions. Seawind [9] is another traffic shaper that has been designed to emulate wireless networks focusing on GPRS networks. It provides a more accurate delay support by specifying allocation, transmission, and propagation delays. By using many instances of traffic shapers it is possible to build large emulation testbeds, for example, Netbed/Emulab [10]. In these large testbeds, various approaches are used to allow the dynamic configuration of a traffic shaper during emulation.

The solution presented in [11] proposed a trace-based emulation system. This emulator mimics a real network on an experimentation one according to previously captured traces. Those traces are then processed to produce an emulation model (i.e., a model made of bandwidth, delays and losses) that will be interpreted by the traffic shaping part of the emulator. This process is known as the ”distillation” process. It results in an accurate and repeatable emulation of what happened in the real network. However, capturing traces can be difficult and expensive in time, for instance, in large-scale networks. Moreover, traces are a snapshot of specific network conditions at some moment in time. For a user, it is nearly impossible to set up any unforeseen conditions.

Several network emulators such as Mobinet [12] have chosen not to rely on an existing traffic shaper but to develop a specific one from scratch. Mobinet is a wireless extension of ModelNet [13] and is composed of two kinds of nodes: the border nodes that will host the virtual nodes where the application and protocols under test are installed and core nodes that will be in charge of the emulation. Mobinet is therefore considered as a centralized emulation solution. To emulate a wireless network, several modules have been implemented: a virtual MAC layer that uses a finite state machine to reproduce the 802.11 b medium access mechanism and a virtual routing layer which uses an implementation of the DSR [6] routing protocol. This solution is similar to the one proposed by JEmu but in Mobinet every module can be replaced if necessary. For example, it would be possible to switch from DSR to AODV [14] if an implementation were available. It has also to be noted that the available bandwidth in this emulator is very impressive and can support up to 49 concurrent traffics of 49 Mbit/s. Nevertheless, node mobility is not supported and the propagation support is very limited only allowing the computation of propagation effects as a distance function. Moreover, DSR is a very light routing algorithm in terms of computation time. Using a more complex routing protocol such as OLSR [15] may have a severe impact on the real-time constraints encountered in emulation.

Finally, some level-3 network emulators are based on real-time discrete event simulation such as NSE [16] (the emulation extension of NS-2) or NCTUns [17]. In those solutions, the real traffic is intercepted by the simulator which computes in real-time the impact of the wireless network on each packet. The main drawback of this solution is the difficulty to respect real time constraints because the more complex the models are, the more computation time is needed. Even with recent optimizations such as those proposed, in [18], NSE is still not scalable and can only support a small set of wireless emulated nodes. For instance in [18], only six wireless nodes, grouped in three pairs, are supported. To further increase performances, the authors investigate the possibility of distributing the real-time simulation process but at the time of writing these solutions have not been evaluated.

The remainder of this paper focuses on level 3 emulation.

3. Problem Statement

This section highlights four identified aspects that are challenging for emulators.

3.1. Accuracy. When performing emulation, the *accuracy* of the emulated conditions should always be considered. The emulated conditions are often meant to reflect actual conditions in a network. The transition from actual conditions to emulated conditions is typically made either by the use of analytical models, simulation or based on traces captured in real networks.

In wireless networks the environment and the behavior of the end user have a strong impact on the observed QoS at the IP level: the current location of the communicating entities and the propagation of the radio signal directly affect delays, losses, and the available throughput that IP packets experience. To emulate wireless networks it is necessary to reproduce these effects as accurately as possible. Dealing with movements of terminal nodes may lead to the use of rather complex models to provide a realistic behavior. For example, [19] proposes the use of a database to predict the next position of a terminal according to its previous location, the time, and its social relations. People do not act the same way during week end or working days. Their movements have to be affected accordingly.

Similarly, the propagation model often needs to be complex in order to provide results that are sufficiently accurate. If the propagation model is too simple, the behavior is not realistic and this can in turn lead to misinterpretation of experimental results. For instance, relying on a uniform independent packet loss model often leads to false results as shown in [20]. The use of accurate propagation models such as those based on Ray Tracing or on Rayleigh or Ricean random distributions [21] helps to reduce this kind of misinterpretations, but at the expense of large amounts of computation time.

The fact that the most accurate models are typically also the most computationally intensive leads to a trade-off between accuracy and scalability when the models are used in a real-time emulation setting. To avoid this trade-off problem, we developed the use of *off-line precomputation of emulation scenarios*, which allows the use of accurate but resource demanding analytical and simulation models.

3.2. Dynamicity. There are several sources that contribute to the evolution of network conditions between a pair of nodes in a wireless network: fast and slow fading of the radio signal, node movement, and varying cross traffic. The timescale of dynamicity is different among these sources, and so is the nature and complexity of the underlying process. The handling of dynamicity in an emulated environment needs to consider both the need for realistic models and the constraints of emulation.

In the previous section, we have proposed the utilization of an off-line precomputation to use realistic but computationally intensive models. This solution is suitable for the accuracy problem but leads to new challenges on a dynamicity standpoint. First of all is the time granularity challenge. An off-line simulator can produce a *dynamic scenario of emulation commands* describing the evolution of the wireless network topology. This dynamic scenario can then be played in real time by a traffic shaper on an emulation testbed but the time granularity used between two updates is critical. Indeed, even if accurate models have been used for mobility and propagation, updating the emulation conditions only every second will decrease their impact on the realism of the emulation. The used time step results from a compromise: the time step has to be small enough to keep the realism of the used models but also has to be big enough so that the traffic shaper is able to apply emulation commands before the next time step.

The second challenge of dynamicity that has to be faced is directly linked to the emulation purpose. A main advantage of emulation is that no traffic model is needed to provide results and evaluate an application. As a matter of fact, it is not always possible to have a traffic model even for a simulator like NS-2: for instance, it is nearly impossible to model the end user behavior of an interactive application; so the resulting traffic model is hardly ever available. To keep the advantage of not being dependent on a traffic model, it is necessary to avoid the use of a traffic model during the off-line precomputation although, in wireless networks, specific problems such as hidden or exposed terminals are related to the traffic. Dealing with these traffic-based problems has to be done in real-time. Two solutions can be considered. First, the simulation process can be done partially in real time as in the EMWin [7] solution. Every node in the experiment can embed a small part of the simulator that will react in real time according to the traffic. The main limitation of this solution is that every node has to be tightly synchronized with the others to ensure the overall consistency of an experiment. Moreover, a dedicated communication protocol has to be set up in order to allow communications among simulator parts. This protocol is used to coordinate the emulation of specific traffic behaviors that involve several nodes simultaneously such as hidden terminals. The second solution to deal with traffic in real-time is to simulate off-line several traffic possibilities and then, on-line, to observe the traffic during the experimentation and to react accordingly. We have chosen this solution because it is less intrusive than the previous one and does not need any time synchronisation between end nodes nor dedicated communication protocol. A more comprehensive description of this solution is provided in Section 4.2.

In both cases, the resulting emulation solution has to be over-provisioned to cope with the real-time constraints and must use a time step small enough between two emulation conditions updates in order to keep the dynamic aspect and the realism of those conditions.

3.3. Repeatability. The third issue that is highlighted in this paper is *repeatability*. Repeatability can be considered from multiple viewpoints. On the highest abstraction level repeatability is the ability to identically reproduce any given experiment setup in order to validate the results of an

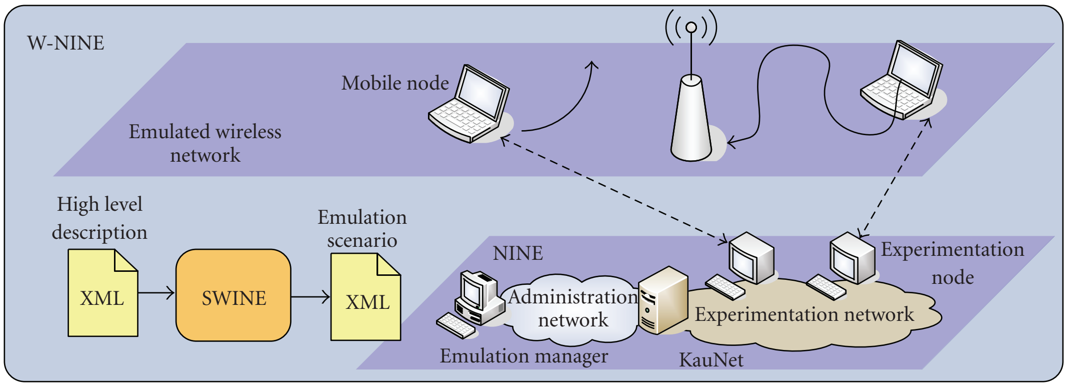


Figure 2: W-NINE Architecture.

experiment performed earlier. Factors that may cause differences between different runs of an identical experimental setup should be minimized. On a lower abstraction level this translates to the amount of control over the experimental environment that is possible to achieve. From a repeatability standpoint, the maximum possible amount of control of the emulated network is desirable. An analysis of the cause for indeterminism in current network emulators gives at hand that one major contributor is the stochastic insertion of packet losses. When an experimental setup contains a noncongestion-related packet loss element, the random positions of these losses will vary between runs and this variance can have considerable impact on the individual experimental results as is shown in [1]. For instance, if a user wants to compare two error control mechanisms such as FEC or hybrid ARQ, it is necessary to ensure that packet losses will be exactly the same in both cases. If losses do not occur on the same packets, it is hard to examine the behavior of the two error control mechanisms in a fully repeatable fashion and it is also harder to conclude which one has the best performance. Similar arguments hold for bit-errors, for instance, in the context of audio/video codec development. To ensure a maximum amount of control over losses and bit errors, *deterministic error positioning* is needed.

3.4. Realism. Realism depends on the three previous points. For example, a high level of realism in a wireless emulation experiment can only be achieved if the emulation is *accurate* enough to produce highly *dynamic* conditions and if the emulator is able to reproduce these dynamic conditions in real time. The *repeatability* also impacts the realism and is highly tied to the design of the emulator. As previously discussed, if an experiment is not fully repeatable, then the accuracy is no longer ensured. To sum up, realism is the goal to strive for but not to the detriment of the controllability of the experiment.

4. W-NINE Architecture

This section describes W-NINE, an emulation platform that addresses the issues of Section 3. As depicted in Figure 2, W-NINE integrates and enhances various existing tools. This integration work was done within the framework of the FP6-IST European Network of Excellence in Wireless Communications (NEWCOM).

W-NINE was designed to address the challenges described previously in Section 3. W-NINE is based on an off-line simulator called SWINE and an emulation platform called NINE. Doing the simulation off-line gives the possibility to use accurate and even realistic models, which are generally time-consuming and cannot be used in real time. To test a protocol or an application, the end user first writes a *high-level description* of the experiment which is processed by SWINE to produce an *emulation scenario* that ensures dynamicity. Then the experiment can be repeatedly run in real time on the NINE platform: the applications and protocols under test are deployed on NINE terminals, while the emulation scenario is played in real time by the emulation manager. The emulation manager periodically sends emulation commands on the administration network to KauNet [22], a Dummynet extension, used in the central router of NINE. KauNet purpose is to, in real-time, constrain all the IP traffic issued by the terminals on the experimentation network thus mimicking the wireless network where applications and protocols under test will be deployed. KauNet was chosen due to its enhancements in terms of repeatability and accuracy as compared to Dummynet. The behavior of these tools is detailed throughout the remainder of this section.

4.1. SWINE: Simulator for Wireless Network. The Simulator for Wireless Network Emulation [23, 24], presented in Figure 3, is an object-oriented Java simulator that produces off-line an *emulation scenario* based on a *high-level description*. Doing this off-line gives the possibility to use accurate and/or realistic models that cannot be used in real time. The emulation scenario will be used in the second stage by KauNet.

The emulation scenario and high-level description are both XML files that must be validated against schemas. In this section, the high-level description file format and SWINE's architecture are presented.

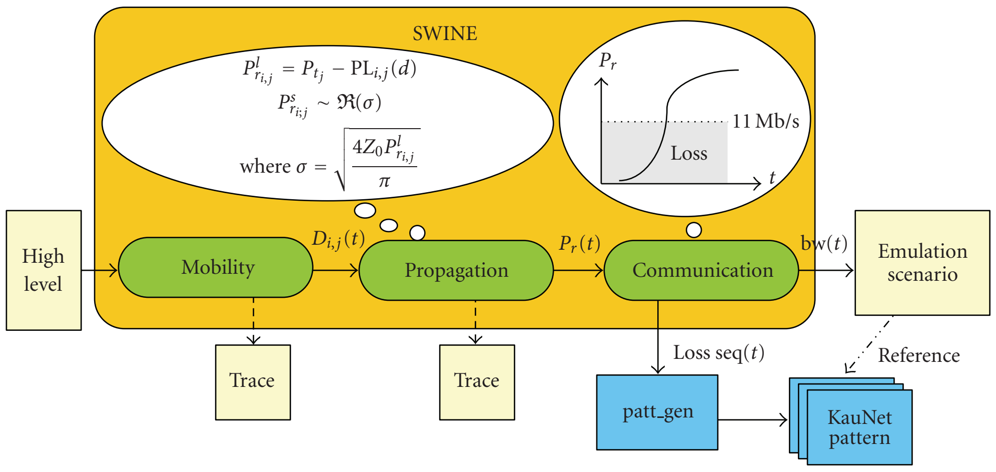


FIGURE 3: SWINE Architecture.

4.1.1. High Level Description File. The High-Level Description (HLD) file describes the experiment to emulate including obstacles and nodes as well as models used to represent, for instance, the propagation conditions or the mobility. This file is written in XML due to its simplicity and to the validation possibilities that are available with this formalism. For this validation purpose, the RELAX-NG compact schema (RNC) formalism [25] defined by OASIS has been selected because it is easier to manipulate than regular XSD schemas due to a structure that is close to the Backus-Naur formalism.

As depicted in Figure 4, an emulation experiment starts with the definition of the kind of network that have to be emulated: an 802.11 Ad Hoc network or a Managed network representing an 802.11 infrastructure network.

Based on the network type, the world where the experimentation takes place is described. The *World* is composed of three main elements: concrete elements (such as obstacles, areas, and nodes), the time information, and the world dimensions. An obstacle is defined by its shape, its position, and possibly its signal absorption capability. An Area allows the introduction of a constant PLR in a geographical area and then is defined by its position and its size. Finally, a *Node*, presented in Figure 5, is mainly composed of the IP information such as its IP address and its gateway but also of its transmission power, its antenna gain, the mobility model that will be used during the simulation to describe its movement and possibly an energy model. Based on the type of network, two kinds of nodes can be considered: mobile nodes and access points which can be viewed as static mobile nodes. This distinction is needed to simulate infrastructure networks. The IP information is provided for the real-time emulation step and is not used directly by SWINE.

The *models* section describes every model used for the computation of the traffic conditions in the wireless network. These models, such as the propagation model (determines the radio signal level received by a mobile node) or the MAC level model (determines the IP throughput available for the end user), are used by every node in the network. Models used to build cells in an infrastructure network or to find potential hidden terminals situations (see Section 4.1.2) are also described in this section. In Figure 6, an example of Pathloss Exponent propagation model [21] description is presented. First, the end user must specify the exponent that will change according to the location of the wireless network and he must also give the reference distance value d_0 and possibly the frequency used by the wireless network and the pathloss value at d_0.

4.1.2. SWINE's Architecture. The main difference between SWINE and classical network simulators such as NS-2, Opnet, or GloMoSim is that SWINE has been designed for an emulation purpose. SWINE does not evaluate a protocol nor an application but it provides the conditions that the traffic will encounter in the target network. Most of classical network simulators use a layered architecture and simulated packets that go through this layered architecture to provide results. In an emulation approach no traffic model is provided in order to evaluate protocols and applications. This specificity has been kept in SWINE's design. Therefore, SWINE computes the best conditions that a node might encountered on a communication link, that is, when it is the only emitting node.

SWINE is a discrete event simulator split in three main steps: *Mobility*, *Propagation*, and *Communication*, each step hosting a number of models with an open architecture that allows the easy addition of new and/or more realistic models.

The *Mobility step* computes all positions of all nodes at every time step of an experiment, using models ranging from the classical random waypoint mobility model to group mobility models such as the Pursue model. All these models can take into account obstacles of the world. Although they are not integrated in SWINE, more complex models such as those which are using social information databases [19] could also be used even though they are time-consuming since there is no time constraint in the off-line preliminary simulation.

```
start = Experiment

Experiment = element experiment {
    element type_of_network {
        attribute id {"Ad-Hoc" | "Managed"}
    }?,
    World,
    element models { NetModel+ }
}

World = element world {
    element time {
        element duration {duration_unit? & real} &
        element step {duration_unit? & real}
    } &
    element origin {distance_unit? & tuple3d} &
    element dimensions {distance_unit? & tuple3d} &
    element obstacles {Obstacle*} &
    element areas {Area+}? &
    element nodes {Mobile+ & AccessPoint*}
}
```

(a) RNC schema

```
<experiment type="Ad-Hoc">
    <world>
        <time>
            <duration unit="s">179.5</duration>
            <step unit="s">0.5</step>
        </time>
        <origin unit="m">
            <x>-2</x>
            <y>0</y>
            <z>-2</z>
        </origin>
        <dimensions unit="m">
            <x>44</x>
            <y>0</y>
            <z>44</z>
        </dimensions>
        <obstacles> ... </obstacles>
        <nodes> ... </nodes>
    </world>
    <models> ... </models>
</experiment>
```

(b) Associated XML example file

Figure 4: High-Level Description File schema.

```
Mobile = element mobile {
    attribute id {xsd:ID} &
    element models {MobilityModel&EnergyModel?} &
    element gain {power_unit? & real} &
    element tx_power {power_unit? & real} &
    element ip_address {text} &
    element ip_mask {text} &
    element gateway {text} &
    element member_of {text} &
    element mapped_on {text}
}
```

(a) RNC schema

```
<mobile id="M1">
    <models> ... </models>
    <gain unit="dB">0</gain>
    <tx_power unit="dB">-13</tx_power>
    <ip_address>192.168.106.1</ip_address>
    <ip_mask>255.255.255.0</ip_mask>
    <gateway>192.168.106.100</gateway>
    <member_of>Emulated WiFi Network</member_of>
    <mapped_on>wnine1</mapped_on>
</mobile>
```

(b) Associated XML example file

Figure 5: An excerpt of the HLD file for nodes.

```
PathLossExponentModel = element stage {
    attribute id {text} &
    attribute class {"swine.models.propagation.PathlossExponentModel"} &
    element exponent {real} &
    element d0 {distance_unit? & real} &
    element frequency {frequency_unit? & real}? &
    element pathloss_d0 {power_unit? & real}?
}
```

(a) RNC schema

```
<stage id="pathloss" class="swine.models.propagation.PathlossExponentModel">
    <exponent>4.6</exponent>
    <d0 unit="m">1</d0>
    <frequency unit="GHz">2.457</frequency>
</stage>
```

(b) Associated XML example file

Figure 6: An excerpt of the HLD file for Pathloss Exponent.

The *Propagation step* uses the positions provided by the mobility step to compute the power of the radio signal received by each node from every other node at every time step of the experiment. Classical propagation models are implemented, including Rayleigh and Rice fading as well as Pathloss Exponent models. Three levels of propagation have been investigated to be as close to reality as possible: large-scale effects that are mainly based on the distance between the sender and the receiver, medium-scale effects resulting of the signal attenuation caused by obstacles between the sender and the receiver, and finally-small scale effects which are caused by the multiple paths that the radio signal takes to reach its destination. By combining these three kinds of effects, it is possible to model propagation in a somewhat realistic way. However, the obtained realism cannot be better than the realism of the combined models. As in the mobility step, more complex and time-consuming models such as ray tracing models [26] could be integrated to enhance the realism of the emulation.

The *Communication step* uses the propagation information to compute the QoS parameters on every link at every time step of the experiment. These QoS parameters form an emulation scenario that can be played by KauNet (see Section 4.3.1). The Communication step computes the maximum available bandwidth on a link with a time step of typically 100 ms as well as KauNet loss patterns with a granularity of 1 milisecond. Based on the IP information, the Communication step is also in charge of partially dealing with traffic based effects such as hidden or exposed terminals, as further discussed in Section 4.2. In future work, the communication step is planned to be extended to more accurately take into account the ad hoc routing by introducing some delays (caused by route error, route discovery, and control messages) on each link according to the routing protocol used.

On a technical standpoint, SWINE relies on two kinds of objects: domain objects and model objects. Domain objects represent the physical elements of the emulated network (wireless nodes, access points, obstacles, etc.) as well as the wireless links. Model objects represent the equations used to compute the network conditions using domain objects. These two kinds of objects are initialized with the provided high-level description file and are then used by the simulation core to fill a link matrix.

The Link Matrix, presented in the UML class diagram in Figure 7, is the key structure in SWINE. In this matrix, the wireless conditions are stored for every unidirectional communications between two nodes at every time step of the experiment. These conditions are composed of the IP Level conditions such as the IP available throughput, delays, and losses but also the propagation informations such as the received signal strength on the receiver side, the signal-to-noise ratio (SNR), the fading, shadowing, and path loss effects. These conditions can also take into account any communication conditions such as a potential hidden terminal on the specified communication link.

To fill this link matrix, several models have been implemented. Figure 8 depicts the model hierarchy that has been used to provide conditions to emulate on the emulation platform. From a generic Model class, four kinds of models have been described. The *Mobility model*, *Propagation model*, and the *Communication model* correspond to the three main steps previously presented and work, respectively at the node level, the link level, and the link matrix level.

In addition to these three main models a fourth one is provided allowing, for example, the simultaneous computation of link level informations and of informations that are related to the whole matrix. Hidden terminals [27] are a good example of such a use. In a hidden terminal situation it is necessary to have a look at the full matrix to find the potential three nodes where a hidden terminal can occurs. For example, in Figure 9, the node $M2$ is in the transmission range of nodes $M1$ and $M3$. Nodes $M1$ and $M3$ are out of range of each other; they cannot sense each other's transmissions. They are said to be mutually hidden. In this case, $M3$ can start sending its packets to $M2$ while $M2$ is still receiving packets from $M1$, which may lead to severe interferences and therefore in losses at the IP level on links $M1 \rightarrow M2$ and $M3 \rightarrow M2$. Both communication links have then to be updated in the communication matrix so that the hidden terminals situation is reproduced during the emulation stage. This traffic-based behavior emulation is detailed in Section 4.2.

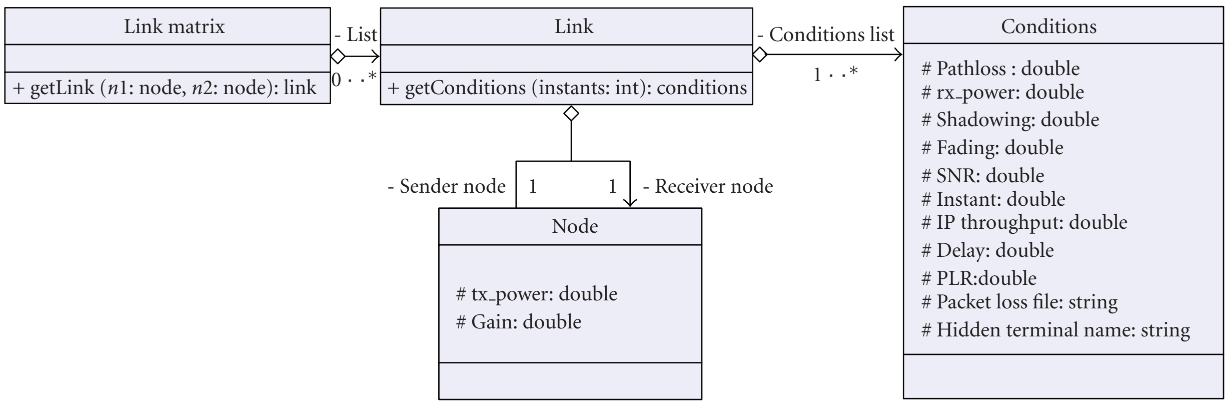


Figure 7: An excerpt of class diagram focusing on the link matrix.

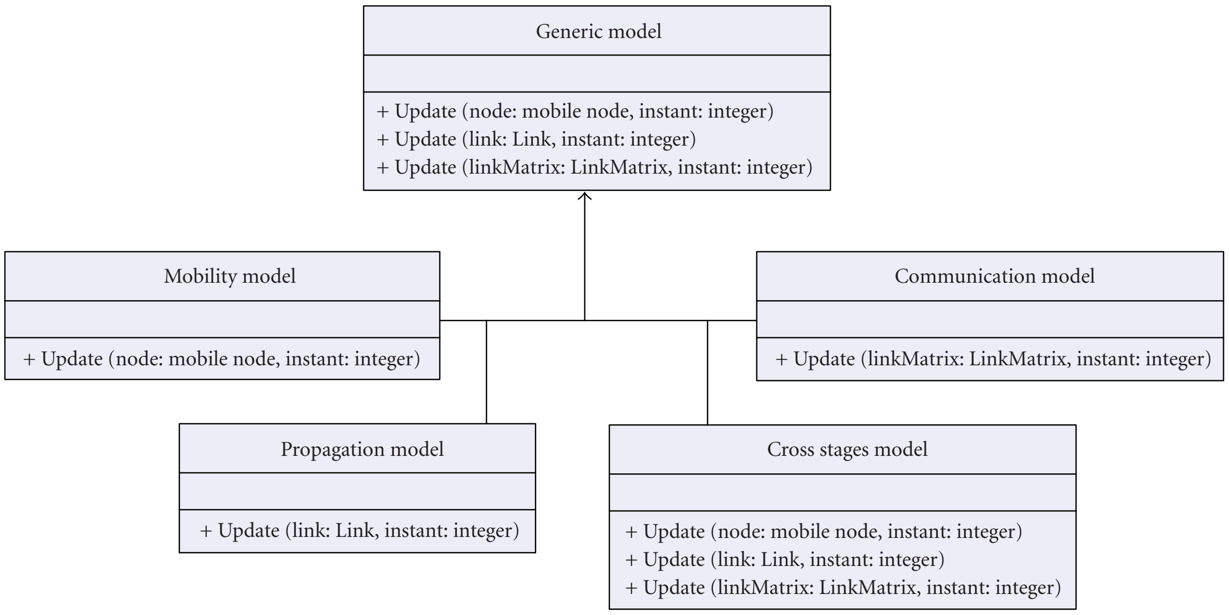

Generic model
+ Update (node: mobile node, instant: integer)
+ Update (link: Link, instant: integer)
+ Update (linkMatrix: LinkMatrix, instant: integer)
Mobility model
+ Update (node: mobile node, instant: integer)
Communication model
+ Update (linkMatrix: LinkMatrix, instant: integer)
Propagation model
+ Update (link: Link, instant: integer)
Cross stages model
+ Update (node: mobile node, instant: integer)
+ Update (link: Link, instant: integer)
+ Update (linkMatrix: LinkMatrix, instant: integer)

Figure 8: An excerpt of the class diagram focusing on models.

4.1.3. Adding New Models. As previously presented, SWINE is an open architecture that mainly relies on model objects. To insert new models in an easy way, the Java class loader is used, allowing the end user to focus on his own model and not on the simulator itself. Inheriting from a predefined model object class, it is possible to write a new model which will then be usable without any recompilation of the simulator core. As there is a validation of the HLD file provided by the end user before simulation, it is also necessary to extend the provided HLD schema so that the new model can be used.

4.2. Emulation of Traffic-Based Behaviors. In wireless networks, instantaneous network traffic has a direct impact on the QoS at the IP level which is far more important than in wired networks. For instance, in Ethernet networks the bandwidth is shared without side effects among nodes which means that if two competing nodes try to send their packets at 10 Mb/s and 2 Mb/s on a 10 Mb/s Ethernet link, the first node will only get around 8 Mb/s. IEEE 802.11 wireless networks do not behave on the same way. In the 802.11 b infrastructure mode, [28] has shown that the maximum available IP data throughput for one node depends on the slowest emitting node. For instance, if a node with a theoretical transmission rate of 11 Mb/s (which corresponds to a maximum IP data throughput of 7.74 Mb/s) wants to emit at its maximum rate while another node belonging to the same cell emits with a 2 Mb/s rate (i.e., an IP data throughput of 1.4 Mb/s), it will be slowed down until it observes an IP data throughput smaller than 1.4 Mb/s. This specificity is directly caused by the CSMA/CA MAC protocol used in 802.11 networks. Other specificities of wireless network are the potential hidden terminals situations previously presented and exposed terminals. As presented in Figure 9, exposed terminals situation occurs when node $M1$ starts a communication with node $M2$ while there is a communication between $M3$ and $M4$.

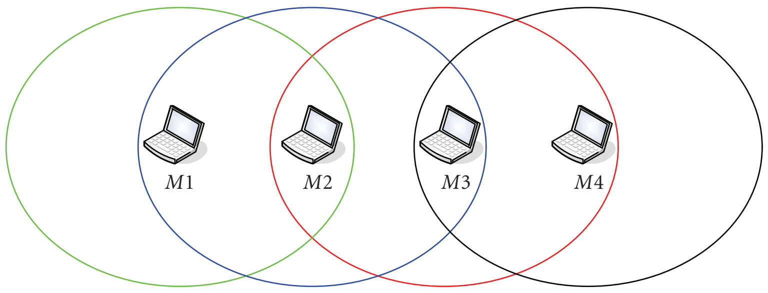


Figure 9: Hidden and exposed terminals situation.

As node $M3$ communicates with $M4$ and is in range of $M2$'s communication, its communication will cause interferences on $M2$ leading to packet losses at the IP level.

Due to the architecture of W-NINE, the emulation process of these wireless network behaviors is split in two stages. The SWINE simulation stage computes the different network traffic possibilities and their impact on wireless communications. SWINE looks for every potential traffic occurrences and generates a specific *choice* in the emulation scenario for each occurrence. According to the 3-step architecture, the complete network topology and available communication links among nodes at each time step are known after the mobility and the propagation steps. During the communication step, SWINE computes the IP parameters that will be reproduced at the emulation stage. First, SWINE's communication step uses the power received computed by the propagation step to determine the corresponding available IP maximum throughput. Second, SWINE investigates the topology to search all potential traffic-based problems. For example in Ad Hoc networks, it looks for every group of at least two nodes that can be hidden from each other. If no potential hidden terminal situation occurs, an emulation scenario with a single choice at each time step is produced. But, if there are potential hidden terminals, SWINE produces an emulation scenario with several choices at each time step. The first choice represents the regular situation when no interference occurs, and the other choices represent situations when at least two hidden nodes try to send packets simultaneously to the same node. These other choices are computed according to the user-specified hidden terminals model in the high-level description file (e.g., a 100% packet loss rate). All of these choices provide a multibranch scenario framework.

During the emulation stage, it is then necessary to observe in real time the network traffic on NINE and to react according to the precomputed choice of the emulation scenario which corresponds to the observed traffic behavior. This mechanism of observation/reaction during an emulation is detailed in Section 4.3.2.

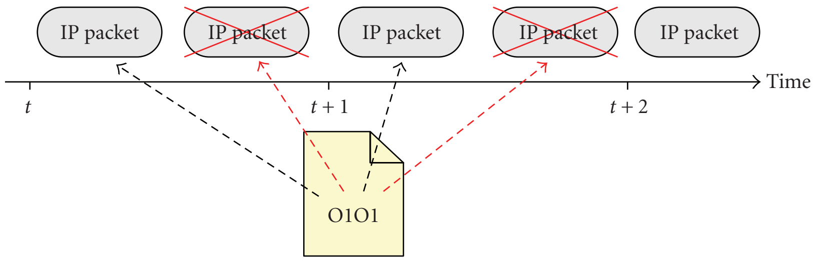
4.3. NINE: Nine Is a Network Emulator. NINE is a fully centralized network emulation platform composed of two dedicated networks: the administration network and the experimentation network. This distinction between networks ensures that the administration traffic does not interact with the experimentation traffic and therefore does not impact the experimentation results. On the administration network, the emulation manager is in charge of the configuration of the whole platform whereas the end user nodes are on the experimentation network. The router emulator is on both networks with one Ethernet network interface on the administration network and several Ethernet network interfaces on the experimentation network. The router emulator is the platform core which means that all packets exchanged during an emulation go through it. This central node hosts the KauNet [22] traffic shaper, responsible of the wireless conditions emulation during an experimentation according to emulation manager instructions.

KauNet addresses fine-grained aspects of dynamicity, repeatability, and accuracy while the emulation manager addresses the dynamicity issue on a larger time scale as well as a part of repeatability with regards to the emulation scenario. Those different aspects will be discussed later on.

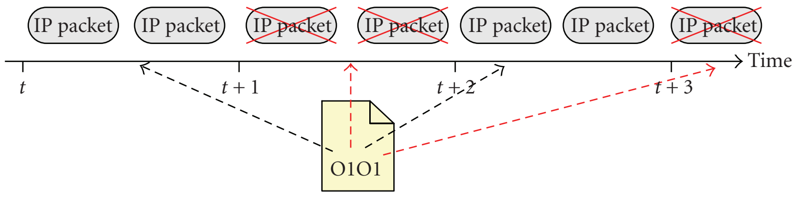
Throughout the remainder of this section, all components of NINE are presented: KauNet, the traffic observers and the emulation manager.

4.3.1. KauNet. KauNet [22] is an extension of Dummynet, developed by the Karlstad University, that provides the ability to accurately place both packet losses and bit-errors at specific locations in a data transfer to examine transport layer protocol implementations and also application layer effects. These loss positions are gathered in specific pattern files that cover only a small period of the experimentation time. Each pattern can be further reused thus ensuring repeatability for the covered period. The losses and errors can be placed either as a function of the amount of packet transferred (data-driven mode) or as a function of time passing (time-driven mode). These two modes are depicted on Figure 10.

When performing evaluation of implementation correctness it is important to be able to have a large degree of repeatability to recreate experimental runs that produce anomalous effects, so that these can be studied in greater detail and then hopefully corrected. As mentioned earlier, the placement of packet losses can have a large impact on the behavior of both transport layer protocols and also application behavior. While the basic protocol mechanisms can be studied by injecting losses in a controlled way using a simulator, this does not help to verify the behavior of actual implementations. KauNet was developed to provide a tight



(a) Data driven mode



(b) Time driven mode

Figure 10: KauNet loss insertion.

control over the accurate placement of losses during a live experiment involving real implementations and real traffic. More technical details can be found in [22].

4.3.2. Traffic Observers. The *traffic observer* (TO) module has been developed to observe in real time the network traffic crossing the NINE experimentation network. The TO is a C++ module hosted on the router-emulator, using a CORBA connection to interact with the emulation manager during the emulation stage. A traffic observer takes no decision; its purpose is to inform the emulation manager when a specific traffic condition occurs on a communication link. Note that if several traffic conditions have to be investigated (e.g., several hidden terminals situations), several TOs are needed.

At the beginning of the experiment, TOs are configured by the emulation manager to observe specific nodes. Then, using feedback mechanisms, a TO sends information relative to the observed traffic to the emulation manager which selects the corresponding precomputed choice of the multibranch emulation scenario at the current time step. For example, if a TO has a couple of nodes under observation in order to determine if they are hidden from each other at time t, it observes the traffic emitted by both nodes and informs the emulation manager when both traffic flows reach the router-emulator during the same time step. The size of this time step and the sets of nodes that must be observed are set by the emulation manager at the beginning of the experiment.

The decision process is centralized in the emulation manager because it is the only node of the administration network that has a complete view of the experiment and of different scenarios. For an administration purpose, this centralized solution is far simpler than distributing the decision process between the emulation manager and TOs. With this centralized solution, KauNet receives update commands only from the emulation manager whereas in a distributed solution a priority scheme between the emulation manager and traffic observers should be set up. The overall deployment process is presented in Figure 11.

A limitation of this solution is that the detection of a situation by traffic observers and the reaction of the emulation manager are not simultaneous. The emulation manager sends KauNet update commands according to the time granularity of the precomputed emulation scenario. That means that in the worst case, KauNet reacts a full-time step after the traffic observers' detection.

4.3.3. Emulation Manager. The emulation manager is a stand-alone Java application in charge of the emulator configuration and of playing the precomputed emulation scenario. The scenario is a natural way to repeat an experiment as often as needed. During an experiment, the emulation manager periodically sends update commands to the emulator in order to make the emulated conditions evolve dynamically.

During NINE configuration, the emulation manager builds every communication link that might be used during the experiment. Communication links are represented by Dummynet pipes in the emulator. According to the precomputed multibranch emulation scenario, the update commands are periodically sent during an experiment by the emulation manager through the administration network to KauNet. These update commands lead to the evolution of communication links' characteristics in terms of bandwidth, delays, and losses over the whole experiment duration. Between two update commands, KauNet is in charge of reproducing the simulated wireless conditions, on a finer time scale which improves accuracy and dynamicity.

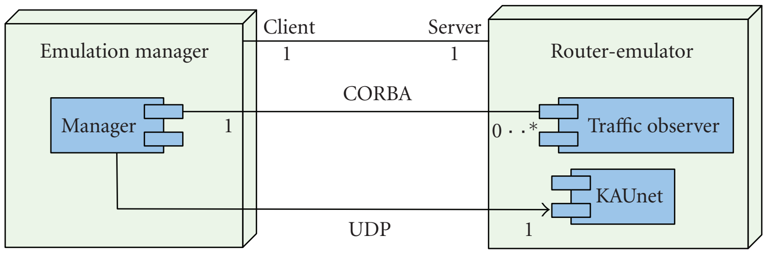


Figure 11: W-NINE deployment diagram.

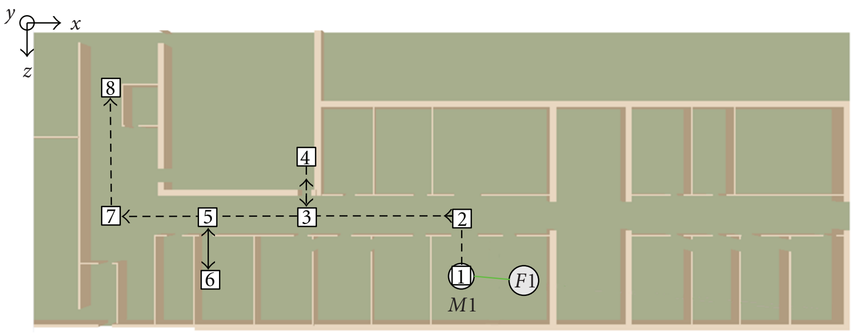


Figure 12: A dynamic experiment.

5. Use Cases

In this section, we highlight with two examples the effectiveness of the methodology and discuss about potential enhancements. First, a simple example presents the KauNet loss pattern generation with SWINE; in this example we compare the emulation result with SWINE simulation results and with the measurements made during a similar real experiment. A second example presents the use of traffic observers to deal in real time with traffic-based behaviors during emulation.

5.1. A Simple Indoor Wireless Communications Experiment

5.1.1. Description of the Experiment. In the experiment illustrated by Figure 12, a mobile user receives a multicast UDP data flow while going through different areas of our offices in Toulouse. At the beginning of the experiment, the user starts close to a stationary sender ($F1$). Then, he follows the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 8$ with a speed of 1 m/s. At $t = 0$ the user starts its movement from his office at position (1); then he enters the corridor (2) and turns to the lecture room (3) which he reaches at $t = 21.3(4)$; he leaves the lecture room and reaches the secretary's office (6) at $t = 35.8$; finally he leaves the secretary's office to go to the exit (8).

While the user is moving, the sender generates a multicast UDP flow with 1472-byte long packets in CBR (constant bit rate) mode. The CBR throughput is 4.19 Mb/s which is the theoretical maximum throughput for a physical transmission rate of 5.5 Mb/s in multicast mode. This 4.19 Mb/s throughput was used so that packet losses can only result from the emulation of wireless losses and not from buffer overflow on the emulation node. Since multicast is used, the link layer will not perform retransmissions.

5.1.2. Live Test Measures. A live test measurement campaign has been performed in our labs, trying to reproduce as close as possible the previous parameters, that is, the mobile trajectory and CBR traffic. Results are depicted on Figure 13 with a granularity of 500 milliseconds.

It can be observed that the PLR increases as the distance increases. This curve will be used as a basis of comparison to evaluate the emulation results provided by W-NINE.

5.1.3. Modeling Packet Losses on a Wireless Link. A key point in wireless networks emulation is the modeling of the environment. We use a *propagation model* to compute the evolution of the PLR over time. A number of propagation models that match many different environments and radio technologies [21] have been developed. We use a combination of a pathloss exponent model and a Rayleigh fading model to provide a reasonably realistic model for an indoor environment (i.e., such a model accounts for both large-scale and small-scale variations of the radio signal). The following experiment shows that a combination of mobility and propagation models can be used to provide more realism when emulating a mobile wireless LAN. Moderately complex models are used here, but much more complex ones could be used to further enhance realism without compromising real-time constraints, since computations based on these models are entirely done off-line *before* emulation time.

The radio signal propagation conditions are described by a combination of pathloss exponent, shadowing, and Rayleigh fading models. The parameters of the pathloss

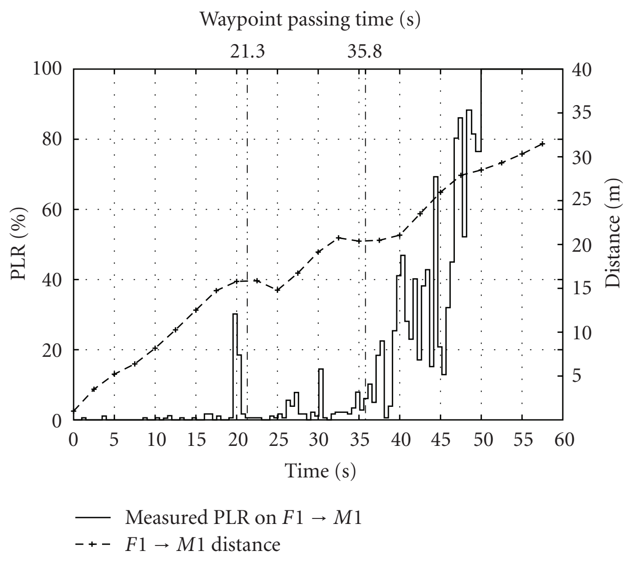


Figure 13: Live test PLR measure.

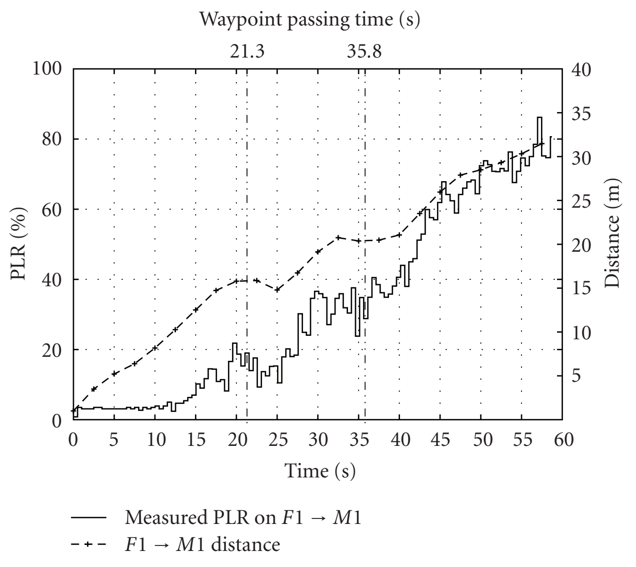


Figure 15: Measured losses on NINE.

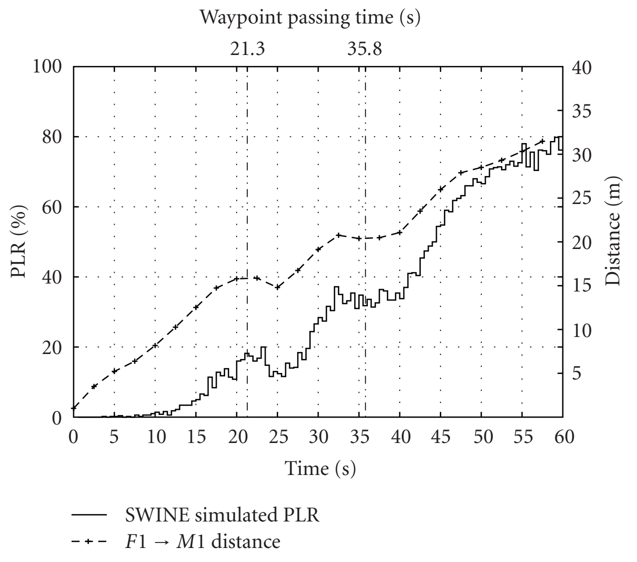


Figure 14: Simulated losses with SWINE.

exponent model are $n = 3.11$, $d_0 = 1\,\mathrm{m}$ and $\mathrm{PL}(d_0) = 37.28\,\mathrm{dB}$. The shadowing model is configured with standard deviation $\sigma = 4.25\,\mathrm{dB}$. Those parameters were estimated using linear regression on the measures of received signal strength (RSS) presented in Section 5.1.2.

5.1.4. Simulation Results. During the simulation stage, SWINE computes the positions of $M1$ as well as the propagation informations. this propagation informations is used to generate small-sized 100 millisecond-long KauNet packet loss patterns with a 1 millisecond granularity. As the experiment is 60 seconds long, 600 KauNet packet loss patterns have been generated. At emulation time, the emulation manager updates the pattern to be played every 100 millisecond. All patterns are used in a data driven way so that the same packet is lost from one run to another.

Figure 14 shows the evolution of the PLR during the experiment. The PLR curve is averaged with a 500 millisecond interval to improve readability. It can be observed that the losses increase as the distance increase. The small variations that can be observed on the PLR curve are due to the fading model effects.

The results are obviously very different between simulation (and therefore emulation) and real measurements. We nevertheless consider that these differences should not be considered as a problem: they simply show that the models used during the simulation stage were not realistic enough. The platform proposed here allows to reproduce with a good fidelity the output of the simulation stage. Realistic emulation therefore depends on the availability of realistic propagation and mobility models as well as MAC and PHY layer models. This topic is extensively discussed in [29].

In the example developed in this section, the propagation model does not take into account the building materials, nor even the number of walls between the sender and the receiver. With such a model, the variations of the simulated signal strength are pessimistic when there are no walls and optimistic otherwise. The litterature proposes more sophisticated models, like the Motley-Keenan model or the COST 231 Multi-Wall Model which will be integrated to the SWINE simulator in future work. On another level, the mobility model used at simulation time supposes a constant speed. The real trajectory of the walking operator during the live tests was not captured, because we did not have accurate indoors positioning tools, which also introduces differences in the results.

5.1.5. Emulation Results. During the emulation stage, the MGEN [30] traffic generator is used to send the desired CBR traffic (4.19 Mb/s) and to collect the packets received by the user. The TRPR tool is then used to generate the PLR curve based on MGEN traces with a 500 millisecond granularity.

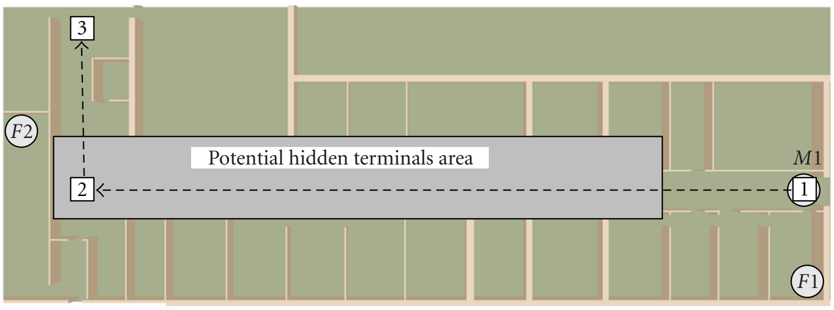


Figure 16: Example of hidden terminal experiment.

```
<date id="t8.4" start="8400" unit="ms">
      <!-- no hidden terminal -->
    <link_update hidden="0" hidden_id="F1-M1-F2">
        <on_link id="F1 to M1" />
        <bandwidth unit="b/s">6870000.0</bandwidth>
        <plr unit="percent">0.0</plr>
        <delay>0</delay>
        <queue_size>8</queue_size>
    </link_update>
      <!-- hidden terminals situation: 100% PLR -->
    <link_update hidden="1" hidden_id="F1-M1-F2">
        <on_link id="F1 to M1" />
        <bandwidth unit="b/s">6870000.0</bandwidth>
        <plr unit="percent">100.0</plr>
        <delay>0</delay>
        <queue_size>8</queue_size>
    </link_update>
    ...
</date>
```

Figure 17: Excerpt of the emulation scenario: two choices for hidden/not hidden traffic conditions.

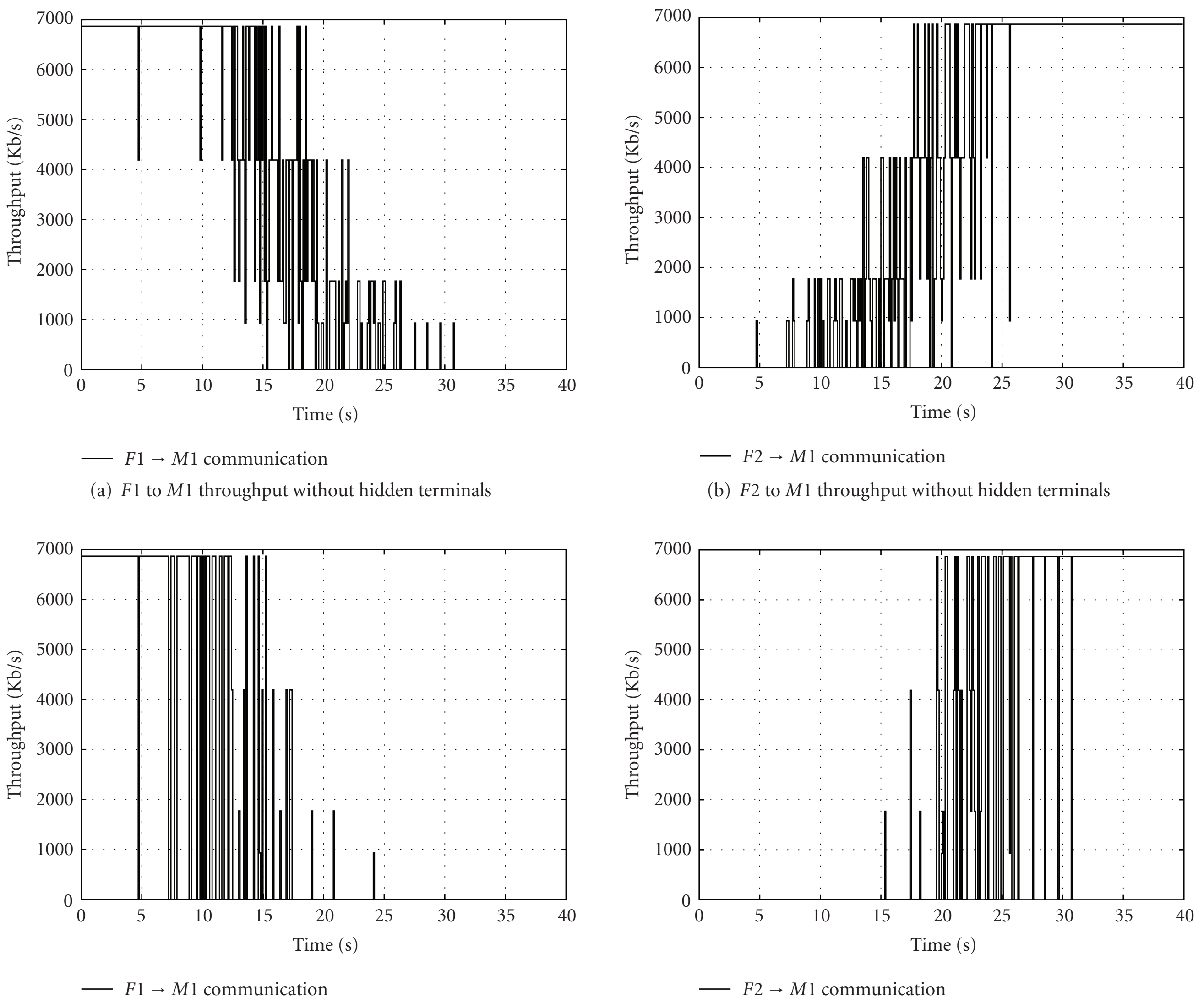
As presented in Figure 15, losses increase with the distance as expected. The PLR observed during emulation is quite similar to the one observed during simulation but shows stronger variations. These variations are due to the packetization by MGEN, which cannot guarantee that the exact same number of packets is sent during each 500 millisecond slot. In these conditions, we consider that the measured PLR is consistent with the simulated one. The performance of KauNet on its own is further examined in Section 6.

5.2. Example of Traffic-Based Emulation. In this use case, we will highlight the usefulness of traffic observers to detect a hidden terminals situation and present the selection process that is made by the emulation manager in order to apply the corresponding emulation command.

5.2.1. A Classical Hidden Terminals Situation. In this example, a mobile node $M1$ and two stationary nodes $F1$ and $F2$ are used. As shown in Figure 16, $M1$ goes through the corridor starting from position 1, located close to $F1$, and stops at position 3 located at the end of the corridors with a speed of 1.5 m/s. The chosen mobility model is simple in order to make the emulation results more readable. $F2$ is located between waypoints 2 and 3.

In this topology, nodes $F1$ and $F2$ are not within range of each other and they are not aware of a potential transmission from the other: this is a typical case where a hidden terminals situation can occur. The propagation of the signal has also to be considered in order to have a reasonably realistic experiment. To deal with propagation a Rayleigh fading model and a Pathloss Exponent model have been used. The parameters of the pathloss exponent model are $n = 5.68$, $d_0 = 1\,\text{m}$ and $\text{PL}(d_0) = 19.97\,\text{dB}$. They are slightly different from the previous experiment because there is no shadowing model explicitly used in this experiment. As in the previous experiment, a multicast UDP flow with 1472-byte long packets in CBR (constant bit rate) mode is used.

The hidden terminals model used during simulation to generate the different choices in the emulation scenario is an ON/OFF model. With this model, when two concurrent signals are received at the same time by the same node, all packets are lost for a time slot duration (100 millisecond).



(a) $F1$ to $M1$ throughput without hidden terminals

(b) $F2$ to $M1$ throughput without hidden terminals

(c) $F1$ to $M1$ throughput with hidden terminals

(d) $F2$ to $M1$ throughput with hidden terminals

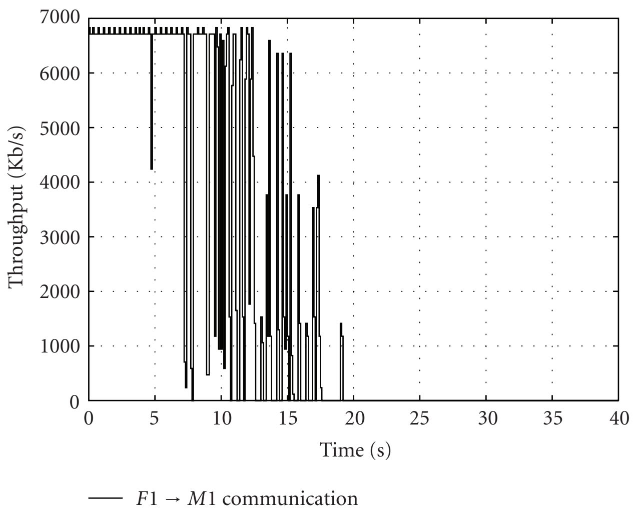
Figure 18: Two simulated possibilities of a hidden terminal scenario.

5.2.2. Simulation Results. The simulation stage generates several choices in the emulation scenario for every potential hidden terminals situation. The generated scenario, presented in Figure 17, details these different choices. The emulation manager is aware that several emulation commands might be sent at a specific moment for a specific communication link and therefore that it has to check for traffic information in order to choose the correct one. A TO called $F1 - M1 - F2$ is then configured to observe the traffic transmitted by nodes $F1$ and $F2$ in order to detect their communication to $M1$.

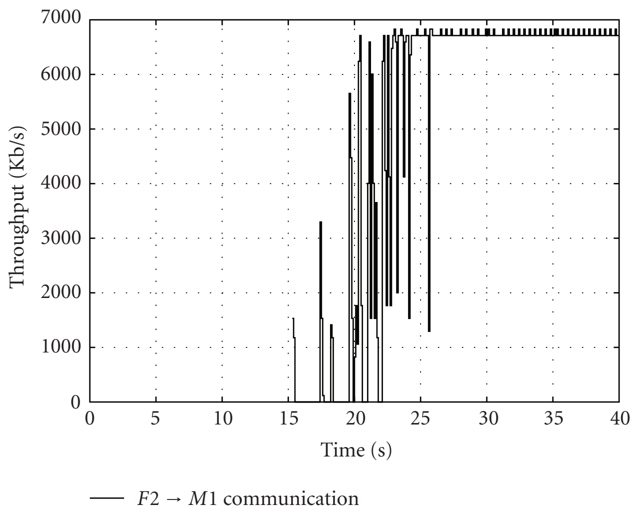
As presented in Section 4.2, SWINE checks the wireless network topology at every time step of the experiment in order to find a topology with potential hidden terminals and generates two choices when an occurrence of hidden terminals is possible: at $t = 8400$ millisecond if the TO detects only one sender, the hidden terminals situation does not happen and so the first choice is applied (PLR = 0%); if the TO detects several senders at the same time, then the second choice is applied and all packets are lost (PLR = 100%). More precisely, this last choice means that if nodes $F1$ and $F2$ simultaneously send packets to the same node $M1$, all of these packets are lost.

Figures 18(a) and 18(b) present the simulation results in terms of IP throughput for communications $F1 \rightarrow M1$ and $F2 \rightarrow M1$ when only one sender is active. For $F1 \rightarrow M1$, the maximum IP throughput gradually decreases until the communication is lost at $t = 26.2$ seconds. The communication $F2 \rightarrow M1$ is possible only after $t = 6.6$ s and the observed IP throughput increases progressively as the node $M1$ is coming closer to $F2$.

Figures 18(c) and 18(d) present the other possible conditions when both senders are active at the same time. We can see that the node $M1$ no longer receives new packets between $t = 15$ s and $t = 20$ s. More generally, both



(a) $F1$ to $M1$ measured throughput



(b) $F2$ to $M1$ measured throughput

Figure 19: Emulation results with a hidden terminal occurrence.

communications are highly degraded with regards to the previous possible conditions.

5.2.3. Emulation Results. To highlight the emulation results, the communications from the senders are started at different moments. Communication $F1 \rightarrow M1$ starts at the beginning of the experiment and stops at $t = 20$ s. The other communication $F2 \rightarrow M1$ begins at $t = 5$ s and stops at the end of the experiment. With this pattern, the hidden terminals situations can be encountered only after $t = 5$ s and so the first choice (no hidden terminals) of the scenario is applied at each time step before this time. After 5 s, the TO running on the emulator detects concurrent traffics from $F1$ and $F2$ on the experimentation network. It informs the emulation manager of the detected hidden terminal situation. The emulation manager then applies the 100% PLR choice at each time step. At $t = 20$ s, as transmission $F1 \rightarrow M1$ stops, there are no more hidden terminals, so the first choice of the scenario is applied to the transmission $F2 \rightarrow M1$ at each time step.

As presented in Figure 19, the obtained results are consistent with our expectations. Until $t = 5$ s, the $F1 \rightarrow M1$ communication has no perturbation and the measured throughput is the same as the simulated one. After $t = 5$ s, the decrease of the IP throughput is considerably larger than for the simulation without hidden terminal. The hidden terminals situation has been detected by the TO and the emulation manager has applied the 100% loss choice between $t = 5$ s and $t = 20$ s. After $t = 20$ s, the IP throughput variations are smaller and are consistent with the simulation of the communication $F2 \rightarrow M1$ without hidden terminals. The TO has detected that the communication $F1 \rightarrow M1$ is over and that the node $F2$ is the only remaining sender. With this detection, the emulation manager has switched from a hidden terminals situation to the regular one where there is no interference.

6. Performance Evaluation

In this section, we measure how accurately KauNet loses packets and we evaluate its performance from the CPU usage, and memory footprint standpoints.

6.1. Setup. The evaluation of accuracy, CPU usage and memory footprint were performed using the same setup. A sender, a receiver, and a KauNet 1.0.0 box were connected to a gigabit Ethernet switch. The three machines used were simple IBM R50e laptops with 768 MB memory, 1.4 GHz processor, and a gigabit Ethernet interface. The sender and receiver run the Linux Ubuntu 8.04 OS, while the KauNet box runs a FreeBSD 7 OS compiled with KauNet support. The unique gigabit Ethernet interface of the KauNet box is configured with two virtual interfaces, one connected to the sender and the other one to the receiver.

6.2. Accuracy of Packet Losses in Time-Driven Mode. To measure how accurately KauNet loses packets in the time-driven mode, we have generated a number of loss patterns and used the iperf [31] traffic generator to measure how many packets were effectively lost by KauNet. The loss patterns were manually generated with different packet loss rates: 1%, 5%, 10%, 25%, 50%, 75% and 99%, and with different durations: 100, 500, and 5000 ms. For each of these 24 packet loss patterns, iperf was used to transmit different constant bitrate (CBR) traffics from the sender to the receiver through the KauNet box, loaded with the pattern under test in time-driven mode. The tested bitrates were 100 kbps, 1 Mbps, and 10 Mbps. For each bitrate, different packet sizes were used: 64, 512, 1024, and 1472-byte long UDP PDUs. Once the bitrate and the packet size were fixed, the sending time is computed so that a significant number of packets goes through KauNet. This significant number is set with a rule of thumb stating that 100 packets are needed to test 100% of the 1-milisecond timeslots for packet loss or no loss, and

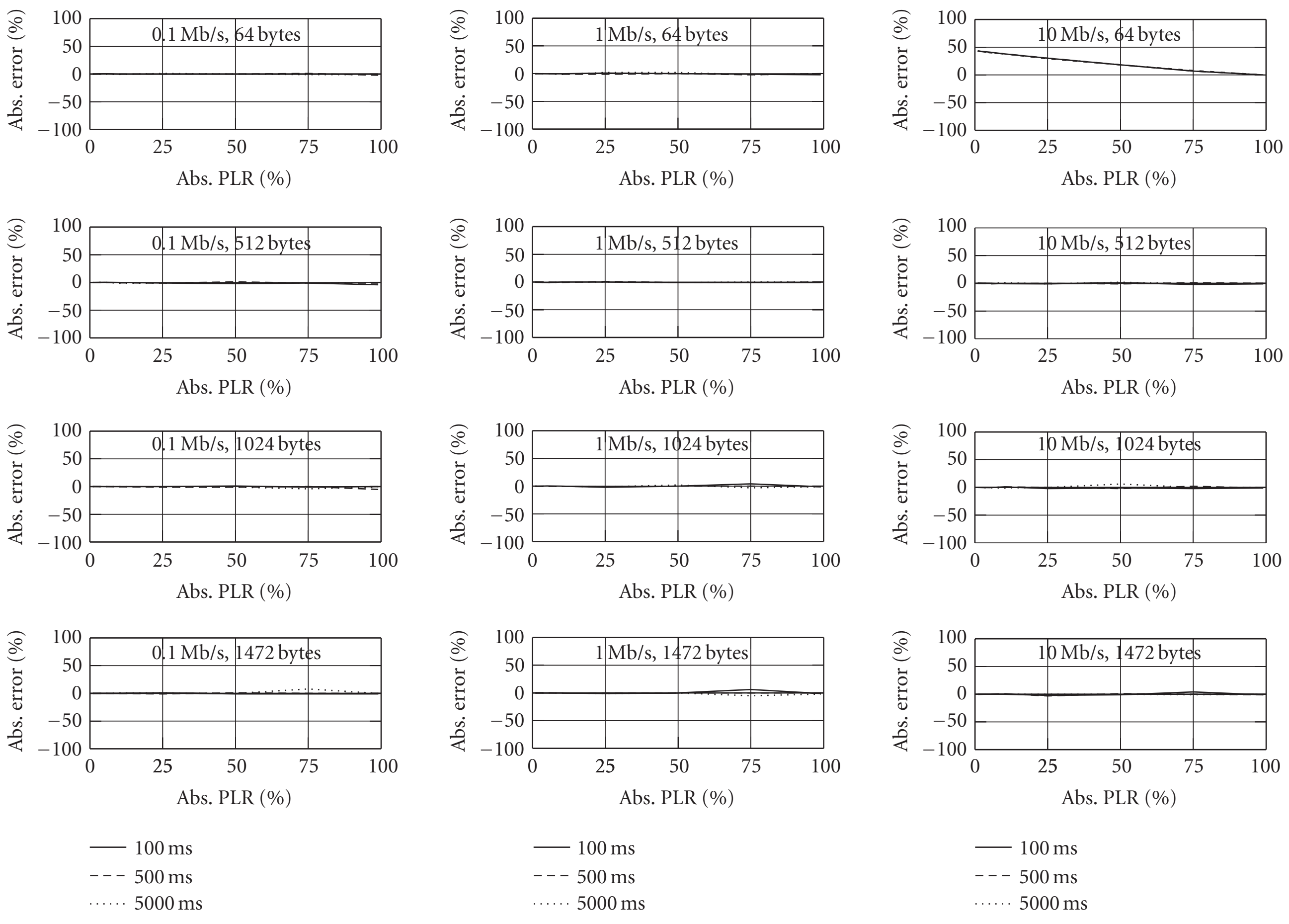


FIGURE 20: Absolute packet loss accuracy error.

that this must be done several times (here 30 times) to gain statistical significance. Hence the sending time is set so that at least 3000 packets are sent and there is traffic during the whole longest 5-second pattern.

To sum up, there are 24 patterns to test, each with 12 different iperf CBR setups. For each pattern generated with PLR_g, iperf provides the observed packet loss rate PLR_o and we measure the absolute error accuracy $E_{abs} = PLR_g - PLR_o$. A negative error indicates that there were more losses than expected.

Figure 20 displays all the absolute accuracy error E_{abs} results. Each subplot has fixed iperf traffic conditions and contains three curves, one for each pattern duration; each curve shows the absolute error E_{abs} observed as a function of the corresponding PLR_g.

The results are very good since in more than 90% of the cases the absolute error is less than 5%. The only case where the relative error gets very high is for the 10 Mbps traffic made of tiny 64-byte packets: in that case, instead of, for example, a 1% PLR we observe a 43% PLR, that is, 43 times more or a 4300% relative error. Under these traffic conditions however, even without any KauNet pattern loaded, iperf observes an erroneous 6% PLR. This is due to the hardware used, which is not powerful enough to correctly manage the 13 packets seen during each millisecond. Future works include running the same tests on a more powerful KauNet box.

To get a more precise idea of the performance of KauNet we also measured the relative accuracy errors $E_{rel} = (PLR_g - PLR_o)/PLR_g$. Figure 21 displays all the corresponding results.

These results are good too: in most cases, the curves are close to 0, which means a negligible relative error. For small values of PLR_g, relative errors of 5% or less are often observed: these error are still acceptable. For instance, a 2% relative error on a 5% PLR gives a 5.1% observed PLR. These small differences are due to the fact that a packet may not exactly fall into the appropriate loss timeslot. More importantly, larger errors of typically 40% are often observed for $PLR_g = 1\%$. This is currently under investigation.

6.3. CPU Usage and Memory Footprint. Running all the above tests takes approximatively 6 hours, taking into account systematic resynchronisation of the 3 machines, pattern loading, and processes starts/stops. During these 6 hours, we measured the CPU and memory usage on the KauNet box using the vmstat system command. The results are illustrated by Figure 22. Most of the time, the CPU load is lower than 5%. Periodic peaks observed correspond to independent periodic tasks that we did not remove. The final

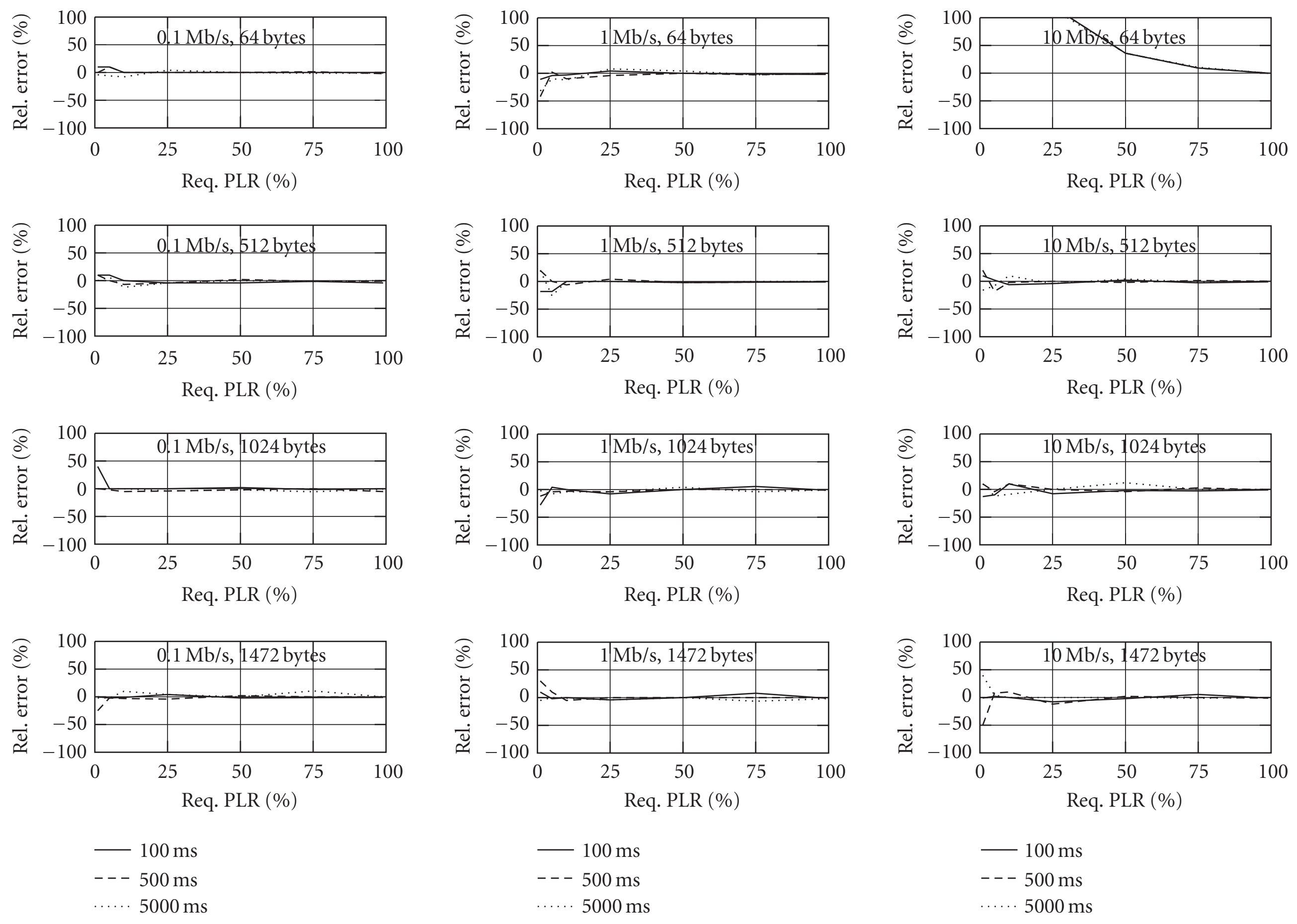


FIGURE 21: Relative packet loss accuracy error.

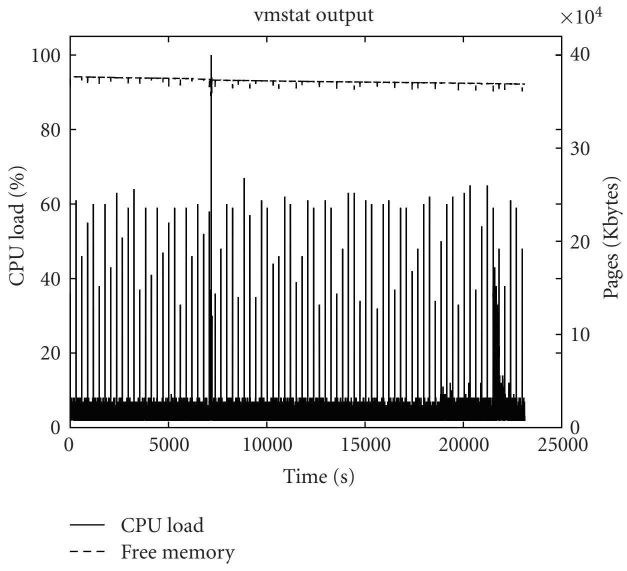


FIGURE 22: CPU and memory usage.

CPU load peak corresponds to the experiments where iperf generates a lot of packets and corresponds to the anomalies described above. Here again, turning to a more powerful KauNet machine can improve the results.

Figure 22 also shows that the amount of free memory is rather stable and that the repeated loading of patterns during the execution of the tests does not cause any large fluctuations in the amount of allocated memory.

7. Conclusion

In this paper we have presented an emulation platform called W-NINE that improves accuracy, dynamicity, and repeatability of the emulated network conditions of a given experiment in order to increase the realism of the experiment. For this purpose, W-NINE is split in two stages. First is the simulation stage based on SWINE, a wireless discrete event simulator which does not need a traffic model to provide results contrary to classical simulators like NS-2. The main purpose of SWINE is not to provide results directly usable by the experimenter but to generate an emulation scenario describing the evolution of conditions (expressed in terms of bandwidth, delays, and losses) in the network to emulate. After the simulation stage is the emulation stage relying on NINE, a fully centralized emulation platform composed of three elements: the emulation manager in charge of the management of the experiment, the KauNet traffic shaper in charge of reproducing the emulated conditions according to emulation commands sent by the emulation manager, and finally traffic observers that check the traffic going through the central node in order to provide feedback to the emulation manager when needed. This traffic observation has been made in order to detect any traffic-related conditions such as hidden terminals for ad hoc networks.

The usability of W-NINE has then been highlighted with two simple case studies: accurate reproduction of simulated conditions over time (case study #1) and traffic-based handling of hidden terminals situations (case study #2). The W-NINE approach allows the designer to test real protocols, applications, and traffic in a real-time environment without sacrificing model accuracy. This provides a unique opportunity to test "black-box" applications and/or protocols under both realistic conditions and limit conditions. Such black-box implementations are at least hard or even impossible (e.g., for copyright reasons) to model using classical simulation platforms.

In future work we plan to integrate the traffic shaping part and the traffic observers in a single module. For this purpose, we work on a new emulator based on finite state machines. With this solution, the medium access protocol can be modeled in a more accurate way.

References

[1] P. Hurtig, J. Garcia, and A. Brunstrom, "Loss recovery in short TCP/SCTP flows," Karlstad University Studies 2006:71, Karlstad University, Karlstad, Sweden, 2006.

[2] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[3] D. Raychaudhuri, I. Seskar, M. Ott, et al., "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC '05)*, vol. 3, pp. 1664–1669, IEEE, New Orleans, La, USA, 2005.

[4] S. Ganu, H. Kremo, R. E. Howard, and I. Seskar, "Addressing repeatability in wireless experiments using ORBIT testbed," in *Proceedings of the 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom '05)*, pp. 153–160, IEEE, Trento, Italy, 2005.

[5] J. Flynn, H. Tewari, and D. O'Mahony, "JEmu: a real time emulation system for mobile ad hoc networks," in *Proceedings of the 1st Joint IEI/IEEE Symposium on Telecommunications Systems Research*, Dublin, Ireland, 2001.

[6] D. B. Johnson, D. A. Maltz, and J. Broch, "The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4," Internet Request for Comments RFC 4728, IETF, 2007.

[7] P. Zheng and L. M. Ni, "EMWIN: emulating a mobile wireless network using a wired network," in *Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia (WoWMoM '02)*, pp. 64–71, ACM, Atlanta, Ga, USA, 2002.

[8] M. Carson and D. Santay, "NIST net: a linux-based network emulation tool," *Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003.

[9] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen, "Seawind: a wireless network emulator," in *Proceedings of the 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB '01)*, pp. 151–166, VDE, Aachen, Germany, 2001.

[10] B. White, J. Lepreau, L. Stoller, et al., "An integrated experimental environment for distributed systems and networks," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pp. 255–270, ACM Press, Boston, Mass, USA, 2002.

[11] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz, "Trace-based mobile network emulation," *Computer Communication Review*, vol. 27, no. 4, pp. 51–61, 1997.

[12] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat, "MobiNet: a scalable emulation infrastructure for ad-hoc and wireless," Tech. Rep., UCSD, San Diego, Calif, USA, 2004.

[13] A. Vahdat, K. Yocum, K. Walsh, et al., "Scalability and accuracy in a large-scale network emulator," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pp. 271–284, ACM, Boston, Mass, USA, 2002.

[14] C. E. Perkins, E. M. Belding-Royer, and S. Das, "Ad hoc on demand distance vector (AODV) routing," Internet Request for Comments RFC 3561, IETF, 2002.

[15] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," Internet Request for Comments RFC 3626, IETF, 2003.

[16] Q. Ke, D. Maltz, and D. Johnson, "Emulation of multi-hop wireless ad hoc networks," in *Proceedings of the 7th International Workshop on Mobile Multimedia Communications (MoMuC '00)*, Tokyo, Japan, 2000.

[17] S.-Y. Wang and Y.-B. Lin, "NCTUns network simulation and emulation for wireless resource management," *Wireless Communications and Mobile Computing*, vol. 5, no. 8, pp. 899–916, 2005.

[18] D. Mahrenholz and S. Ivanov, "Real-time network emulation with ns-2," in *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT '04)*, pp. 29–36, Budapest, Hungary, 2004.

[19] M. Musolesi, S. Hailes, and C. Mascolo, "An ad hoc mobility model founded on social network theory," in *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '04)*, pp. 20–24, ACM, Venice, Italy, 2004.

[20] J. Lacan and T. Pérennou, "Evaluation of error control mechanisms for 802.11b multicast transmissions," in *Proceedings of the 2nd International Workshop on Wireless Network Measurement (WINMee '06)*, IEEE, Boston, Mass, USA, 2006.

[21] T. Rappaport, *Wireless Communications Principles and Practice*, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 2002.

[22] J. Garcia, E. Conchon, T. Pérennou, and A. Brunstrom, "KauNet: improving reproducibility for wireless and mobile research," in *Proceedings of the 1st International Workshop on System Evaluation for Mobile Platforms (MobiEval '07)*, pp. 21–26, San Juan, Puerto Rico, USA, 2007.

[23] T. Pérennou, E. Conchon, L. Dairaine, and M. Diaz, "Two-stage wireless network emulation," in *Proceedings of the Workshop on Challenges of Mobility Jointly with the 18th IFIP World Computer Congress (WCM '04)*, pp. 181–190, Toulouse, France, 2004.

[24] E. Conchon, T. Pérennou, and M. Diaz, "Feedback based solutions to emulate hidden terminals in wireless networks," in *Proceedings of the International Conference on Software, Telecommunications and Computer Networks (SoftCOM '05)*, Split, Croatia, 2005.

[25] J. Clark, J. Cowan, and M. Murata, "RELAX NG Compact Syntax Tutorial," 2003, http://www.relaxng.org/.

[26] A. Schoonen, “Designing wireless indoor radio systems with ray tracing simulators,” Tech. Rep., Eindhoven University of Technology, December 2003.

[27] F. A. Tobagi and L. Kleinrock, “Packet switching in radio channels—part II: the hidden terminal problem in carrier sense multiple-access modes and the busy-tone solution,” *IEEE Transactions on Communications*, vol. 23, no. 12, pp. 1417–1433, 1975.

[28] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, “Performance anomaly of 802.11b,” in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM ’03)*, vol. 2, pp. 836–843, San Francisco, Calif, USA, 2003.

[29] E. Ben Hamida, G. Chelius, and J. Gorce, “On the complexity of an accurate and precise performance evaluation of wireless networks using simulations,” in *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM ’08)*, pp. 395–402, ACM, Vancouver, British Columbia, Canada, 2008.

[30] NRL/PROTEAN, “MGEN: The Multi-Generator Toolset,” http://cs.itd.nrl.navy.mil/work/mgen/index.php.

[31] NLANR/DAST, “IPERF Project,” http://sourceforge.net/projects/iperf/.