Wireless Mesh Network Simulation Framework for OMNeT++

CREATE-NET TECHNICAL REPORT CN-TR-200700016

2007

Authored by: Tinku Rasheed

Wireless Mesh Network Simulation Framework for OMNeT++

Abstract

In this report, we introduce a new scalable wireless mesh network simulation framework which provides a powerful simulation framework and concise modeling chain for Wireless Mesh Networks. The mesh simulation framework, which is still in development and implemented in OMNeT++, already provides a solid base of available models and implementations integrated into the simulation framework environment, including models for mobile environments, nodes, radio propagation models for multiple signal dimensions, physical layer models for modulation, coding and diversity receivers as well as an extensive library of MAC and network protocols. The major objective of the simulation framework development is in the integration of the scattered wireless models in the OMNeT++ simulator into a common framework for effective simulation of WMNs. Thus, the simulation framework profits from the rich experience gathered from the various wireless simulation frameworks of OMNeT++. A modular design enables the straightforward definition of complex scenarios as well as the easy integration of new models and protocol implementations. It is, therefore, our hope that the mesh network simulation framework, once completely developed, and made available to the community, can provide a clear structure and its extensive modeling base and the ongoing development of the integrated simulation-emulation environment can motivate researchers to contribute to the maturity of the mesh network domain from a simulation perspective. Apart from that, the emulation framework development, which is currently in progress within the framework of the mesh simulation toolkit, to interface real networks with the simulations, will enable the WMN research community and the practitioners alike to simulate and parallel run real WMN networks, and to validate the performance, thus bridging the simulation-real network performance gaps.

Contents

CREATE-NET TECHNICAL REPORT CN-TR-200700016

Introduction

Wireless Mesh Networks (WMN) has become one of the key technologies for providing increased network coverage of Internet infrastructures. The simple cost-efficient deployment with self-configuration facilities makes them a valuable alternative to wired networks to increase the network coverage. Therefore, WMNs are in the focus of current research. Several research and city WMNs already exist [1][2][3] and WMNs are evolving from pure research networks to carrier-grade communication infrastructures, which requires extensive pre-deployment testing.

The development process in WMNs is typically split into evaluations by simulations and testing a real prototype in a test-bed. First, the protocols and architectures are implemented and evaluated in a network simulator. Afterwards, a prototype is implemented on the target platform such as Linux and tested inside a test-bed before deployment in the real network. Simulation provides most flexibility in testing. Different and large scale experiments as well as experiments with mobility of devices and users are possible. It provides the best way for testing and debugging the functionality of the approaches. But unfortunately, simulation models cannot cover all influences of the operating system, the network stack, the hardware, and the physical environment due to complexity constraints. Therefore, the transition from the simulation models to the deployable solution remains challenging. Testing the prototype in a test-bed during the implementation process is time-consuming, costly, and very limited. Due to economical reasons, the scale of the testbeds is limited and they are often not deployed in isolated environments, which limit reproducibility. Interferences with existing network are possible and irrepressible, which makes debugging of new protocols very challenging. Furthermore, the number of test topologies is limited and mobility tests are impracticable.

Moreover, WMNs provide an enhanced testing challenge compared to simple wireless access networks. They support mobile users and high-throughput applications. Their architecture contains self-configuring and self-healing mechanisms, which have to be included in the tests. The cross-layer interactions have to be tested in a controlled environment without any irrepressible influences. Moreover, the tests have to cover time and delay aspects of the real network stack. All these tests cannot be fully done in simulations, but also in a test-bed they are difficult to be performed. We therefore propose to emulate the physical medium to gain more control in the development process.

The authors of [4] validated the wireless model in the network simulator ns-2 by comparing measurements of a real network setup with an emulated and simulated network. They concluded that with a proper parameterization the simulation model can approximate the real network, but some aspects like delays introduced by the hardware and the

operating system cannot be considered in the simulation. Therefore, their emulated network provides results that match the real measurement more accurately than the simulation. The approach presented in [5] tries to integrate the behavior of the real network stack and the operating system into the testing process by using virtualized hosts connected through an emulation framework. The virtual hosts are running a L4 microkernel on top of a real-time kernel. To integrate the wireless network behavior, the hosts are connected by the 802.11b network emulator MobiEmu [6]. Although the authors praised the low hardware requirements of their approach, they did not publish any results about the accuracy of their setup.

JiST/MobNet [7] provides a comprehensible Java framework for simulation, emulation, and real world testing of a wireless ad-hoc network. It allows running of the same tests independently of platform and abstraction level. MobNet is a wireless extension on top of the Java in Simulation Time (JiST) simulator. The drawback of this approach is that most communication software and network protocol stacks are not written Java and therefore afterwards a further transition to an embedded system may be necessary. Another approach for testing real implementations in a very flexible network is provided by the ORBIT testbed [8]. It provides a configurable indoor radio grid for controlled experimentation and an outdoor wireless network for testing under real-world conditions. The indoor radio grid offers a controlled environment as an isolated network, in which background interferences can be injected. Although the 20 x 20 grid of nodes offers a large variety of different topologies, it can be too restricted and mobility tests are even more limited. Furthermore, the scarce ORBIT resources may be not available for all experiments.

UMIC-Mesh [9] is a hybrid WMN testbed. Besides a testbed with real wireless mesh nodes, UMIC-Mesh provides virtual nodes by using XEN virtualization. The virtual nodes are interconnected by a combination of the advanced networking features of the Linux kernel. This includes packet filtering for controlling the communication between the nodes. The virtual network is only intended for software development and functionality validation. Therefore, the behavior of the wireless medium is not modeled in this approach. In order to cope with the problem of a simulator overload during network emulation, the authors of [10] introduce the concept of synchronized network emulation. They replace real hosts with virtualized hosts using XEN. A central synchronizer component then controls the time flow of the virtual hosts by an adapted scheduler for XEN. It keeps them synchronized with the network simulator OMNeT++. The integration of real network stacks inside a network simulator provides another mechanism for testing complex protocol behavior. OppBSD [4] integrates the TCP/IP stack of FreeBSD in the network simulator OMNeT++.

Our contribution as part of the WING project is to design and develop a general and open framework to perform scalable wireless mesh network simulations. The framework might be useful for other researchers as well and the implementation of mesh networks from scratch is a tedious and error prone task. We therefore present a combined framework incorporating various simulation frameworks developed for wireless and mobile simulations in OMNeT++. The simulation framework is expected to provide detailed models and protocols as well as a supporting infrastructure. There are several criteria to be addressed when designing a generic simulation framework for a wireless networking environment. In a simulation, only relevant parts of the real world should be reflected, such as obstacles that hinder wireless communication. When nodes move, their influence on other nodes in the network varies. The simulator has to track these changes and provide an adequate graphical representation. For wireless simulations, movements of objects and nodes have an influence on the reception of a message. The reception handling is responsible for modeling how a transmitted signal changes on its way to the receivers, taking transmissions of other senders into account. The experimentation support is necessary to help the researcher to compare the results with an ideal state, help him to find a suitable template for his implementation and support different evaluation methods. Last but not least, a rich protocol library enables researchers to compare their ideas with already implemented mesh network protocols. The simulation framework is managed by a network analysis module which musters the

performance data obtained from the simulations. The framework also implements support to overcome current limitations in the OMNeT++ simulator to perform mesh network simulations by adding new features to the latest available simulator version.

We opted to choose OMNeT++ as the simulator for our implementation and evaluation since OMNET++ is user friendly, open-source and currently, the best open source simulator available for modeling and developing wireless network simulations. The generic simulation framework is assembled by combining the approaches of several existing simulation frameworks into one: the mobility support, connection management, and general structure is taken from the Mobility Framework (MF) module in OMNeT++; the radio propagation models are taken from the ChSim channel simulator; and the protocol library is taken from the MAC simulator and the Mobility Framework. In this technical report, we discuss the OMNET++ simulator, and present the initial architecture of the scalable wireless mesh network simulation environment. We also discuss the evaluation of the performance of a routing protocol in the scalable simulation environment.

The Discrete Event Simulation System – OMNeT++

This chapter serves as a short presentation of the main features of OMNeT++.

Introduction

OMNeT++ is a discrete event simulator based on C++, is highly modular, well structured and scalable. It provides a basic infrastructure wherein modules exchange messages. The name OMNeT++ stands for Objective Modular Network Testbed in C++. It has an open-source distribution policy and can be used free of charge by academic research institutions. It runs on Windows and Unix platforms, including Linux, and offers a command line interface as well as a powerful graphical user interface. The simulator can be used, for instance, to model communication and queuing networks, multiprocessors and other distributed hardware systems as well as to validate hardware architectures.

Modeling Concept

An OMNeT++ model consists of hierarchically nested modules, which communicate by passing messages to each other. OMNeT++ models are often referred to as networks. The top level module is the system module. The system module contains sub-modules, which can also contain sub-modules themselves. The depth of module nesting is not limited; this allows the user to reflect the logical structure of the actual system in the model structure.

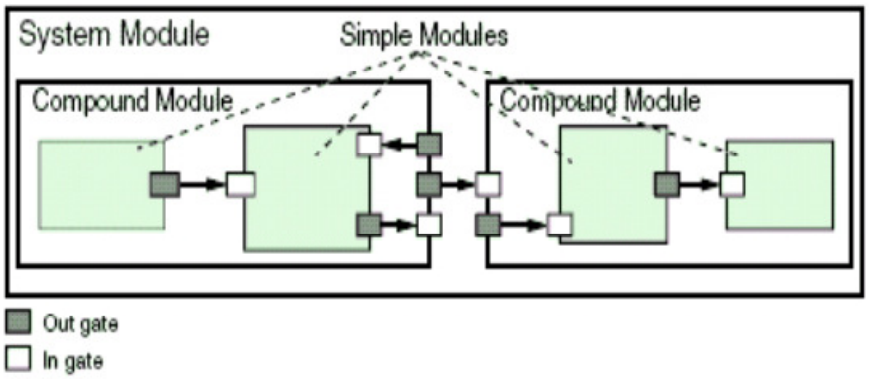

FIGURE 1: OMNET++ MODULE HIERARCHY

The model structure is described with OMNeT's NED language. Modules that contain sub-modules are termed compound modules, as opposed to simple modules which are at the lowest level of the module hierarchy. Simple modules contain the algorithms in the model. The general OMNeT++ module hierarchy is shown in Figure 1. The user implements the simple modules in C++, using the OMNeT++ simulation class library.

Modules communicate by exchanging messages. In an actual simulation, messages can represent frames or packets in a computer network, jobs or customers in a queuing network or other types of mobile entities. The local simulation time of a module advances when the module receives a message. The message can arrive from another module or from the same module (self messages are used to implement timers).

Gates are the input and output interfaces of modules; OMNeT++ supports only simplex (one-directional) connections, so there are input and output gates. Messages are sent out through output gates and arrive through input gates. Due to the hierarchical structure of the model, messages typically travel through a series of connections, to start and arrive in simple modules. Such series of connections that go from simple module to simple module are called routes. Compound modules act as "cardboard boxes" in the model, transparently relaying messages between their inside and the outside world. Connections can be assigned three parameters, which facilitate the modeling of communication networks, but can be useful in other models too: propagation delay, bit error rate and data rate, all three being optional. One can specify link parameters individually for each connection, or define link types and use them throughout the whole model.

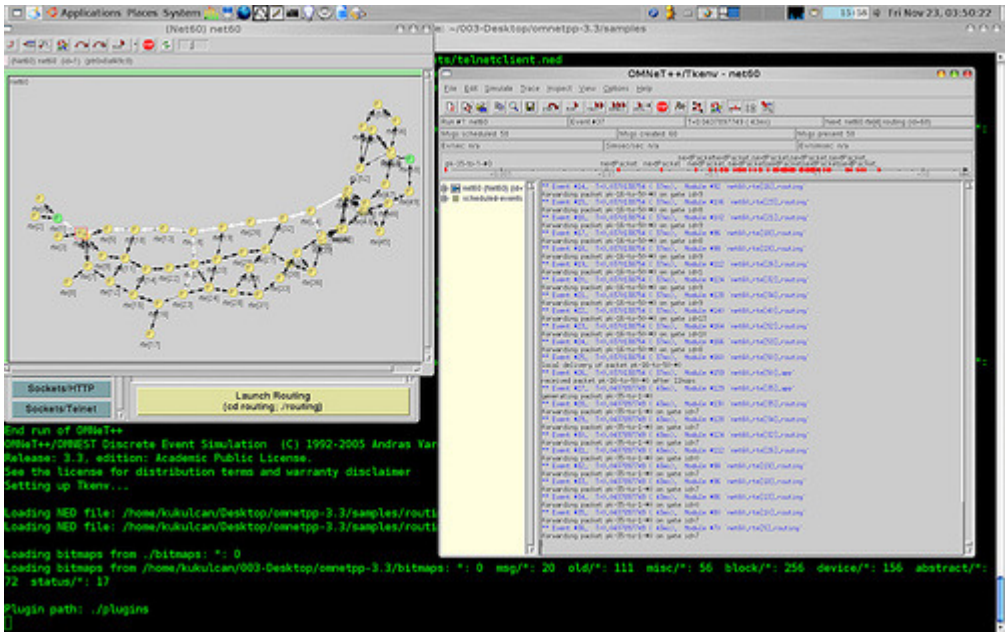

FIGURE 2: OMNET++ SNAPSHOT

The simple modules of a model contain algorithms as C++ functions. The full flexibility and power of the programming language can be used, supported by the OMNeT++ simulation class library. The simulation programmer can choose between event-driven and process-style description, and can freely use object-oriented concepts (inheritance, polymorphism etc.) and design patterns to extend the functionality of the simulator. In Figure 2, a snapshot of the OMNeT++ graphic user interface is provided.

Basic parts of an OMNeT++ Model

An OMNeT++ model physically consists of the following parts:

- NED language topology description(s)
- Message definitions
- Simple modules implementations and other C++ code

To build an executable simulation program, you first need to translate the NED files and the message files into C++, using the NED compiler (nedtool) and the message compiler (opp msgc). NED files can also be loaded dynamically, in which case they don't need to be compiled beforehand. After this step, the process is the same as building any C/C++ program from source.

Interesting features

Below are some of the interesting features offered by the OMNeT++ simulator:

- ✓ Pseudorandom generators.
- ✓ Flexibility.

- ✓ Programming model.
- ✓ Model management.
- ✓ Support for hierarchical models.
- ✓ Debugging and tracing.
- ✓ Documentation.
- ✓ Large scale simulation.
- ✓ Parallel simulation.
- ✓ Experiment specification.

Comparison with other simulators

Available Models Non-commercial simulation tools cannot compete with some commercial ones (especially OPNET) which have a large selection of ready-made protocol models. OMNeT++ is no exception, it clearly lacks models, also compared with non-commercial tools such as ns-2 (but it has to be considered that OMNeT++ is a rather new tool, it was originally released in 1999). On the other hand OMNeT++ provides a larger variety of models (that allows the user to simulate more than just communication networks) as compared to ns, which mainly provides TCP/IP centered models.

Model Management The OMNeT++ simulation kernel is a class library, i.e. models in OMNeT++ are independent of the simulation kernel. The user writes his components (simple modules) like using any other class library, and generates the executable by compiling and linking them against the simulation library. This means that there is no need to modify the OMNeT++ sources (this enforces reusability). ns-2 tends to be monolithic: to add new models to it, one needs to download the full source and modify it a bit, copy files to specific locations, add constants in other files etc.

Reliability As a matter of fact, models provided with simulation tools are often not validated. This also applies to OMNeT++. A good example is the TCP implementation in the INET framework for OMNeT++ (more on that later on). This is a general problem of non-commercial tools: anybody can contribute, but nobody gives any guaranty. Moreover, some models are still under development and therefore represent simplified versions of what they are intended to model.

Network Topology Definition Network simulation tools naturally share the property that a model (network) consists of "nodes" (blocks, entities, modules, etc.) connected by "links" (channels, connections, etc.). Many commercial simulators have graphical editors to define the network; however, this is only a good solution if there is an alternative form of topology description (e.g. text file) which allows one to generate the topology by program. On the other hand, most non-commercial simulation tools do not provide explicit support for topology description: one must program a "driver entity" which will boot the model by creating the necessary nodes and interconnecting them (e.g. in ns-2 the OTcl scripting language is used). Finally, a large part of the tools that support explicit topology description account flat topologies only. OMNeT++ probably uses the most flexible method: it has a human-readable textual topology description format (the NED language) which is easy to create with any text-processing tool (perl, awk, etc.), and the same format is used by the graphical editor. It is also possible to create a "driver entity" to build a network at run-time by program. Moreover, OMNeT++ also supports sub-module nesting without limitations on the depth of nesting.

Configuration of Simulation Runs Parameters of a simulation experiment are written in the omnetpp.ini, this strongly enforces the concept of separating the model from experiments. Models and experiments are usually seriously interwoven in ns-2: parameters are usually embedded in the Tcl script and thus are difficult to edit.

Debugging C++-based simulation tools rarely offer much more than the printf()-style debugging process; often the simulation kernel is also capable of dumping selected debug information on the standard output. OMNeT++ goes a different way by linking the GUI library with the debugging/tracing capability into the simulation executable. This architecture enables the GUI to be very powerful: every user-created object is visible (and modifiable) in the GUI via inspector windows and the user has tight control over the execution. To the author's best knowledge, the tracing feature OMNeT++ provides is unique among the C++-based simulation tools and is comparable to enterprise solutions available with simulators like that of OPNET. In addition, this property makes OMNeT++ an excellent tool for demonstrational or educational purposes.

Performance This is a particularly interesting issue with OMNeT++ since the GUI debugging/tracing support involves some extra overhead in the simulation library. Simulating large networks results in unacceptable performance most of the time. But this is also a big problem with other popular simulators such as ns-2.

Mesh Network Simulation Framework

In this section, we discuss the design and development of the open framework to perform scalable wireless mesh network simulations. We present a combined framework incorporating various simulation frameworks developed for wireless and mobile simulations in OMNeT++. The simulation framework provides detailed models and protocols as well as a supporting infrastructure.

There are several criteria to be addressed when designing a generic simulation framework for a wireless networking environment. In a simulation, only relevant parts of the real world should be reflected, such as obstacles that hinder wireless communication. When nodes move, their influence on other nodes in the network varies. The simulator has to track these changes and provide an adequate graphical representation. For wireless simulations, movements of objects and nodes have an influence on the reception of a message. The reception handling is responsible for modeling how a transmitted signal changes on its way to the receivers, taking transmissions of other senders into account. The experimentation support is necessary to help the researcher to compare the results with an ideal state, help him to find a suitable template for his implementation and support different evaluation methods. Last but not least, a rich protocol library enables researchers to compare their ideas with already implemented mesh network protocols. The simulation framework is managed by a network analysis module which musters the performance data obtained from the simulations. The framework also implements support to overcome current limitations in the OMNeT++ simulator to perform mesh network simulations by adding new features to the latest available simulator version (currently undergoing implementation and is not included in this report). The generic simulation framework is assembled by combining the approaches of several existing simulation frameworks into one: the mobility support, connection management, and general structure is taken from the Mobility Framework (MF) module in OMNeT++; the radio propagation models are taken from the ChSim channel simulator; and the protocol library is taken from the MAC simulator and the Mobility Framework. In this technical report, we discuss the OMNeT++ simulator, and present the initial architecture of the scalable wireless mesh network simulation environment. We also discuss the evaluation of the performance of a routing protocol in the scalable simulation environment.

General Framework Structure

The simulation framework structure can be divided into two parts: the basic framework and the protocol library. The basic framework provides the general functionality needed for almost any wireless simulation, such as connection management, mobility, and wireless channel modeling. The protocol library complements the basic framework with a rich set of standard protocols, including mobility models. In order to have clearly defined interfaces between the basic framework and the protocol library, there is a basic module which is implemented in OMNeT++ to manage these interactions. Adopting such an approach makes it easy to implement new protocols while facilitating re-usability of the framework. The network contains entities which are wirelesss mesh routers communicating with each other. Different types of entities can be specified, such as Mesh Access Points (APs) and terminals. Additionally, a node can have additional communication capabilities like multiple radio interfaces of IEEE 802.11 and also, interfaces other than WiFi, e.g. Bluetooth (heterogeneous scenario). We are presently working to implement this additional feature. There is a *ConnectionManager* module which is responsible for dynamically managing the connections between interfering nodes. It knows the position of all the mesh routers. The *ConnectionManager* can query the *ObjectManager* module to interact with the mobile stations. In general, the simulation framework is designed to support multiple connection managers, responsible for different radio frequency ranges. The mobility module is responsible for the movements of a node or an object. Different mobility models available with the framework are described in Section 5.3. The arp module handles the Address Resolution Protocol (ARP), i.e. the translation between network and MAC addresses. There is a utility module is derived from the blackboard module introduced in the Mobility Framework [10]. It has two main tasks: Firstly, it provides a general interface for collecting statistical data of a simulation. Using the utility module for statistical data collection only has minimal impact on the performance of the simulation and leaves full flexibility for different analysis methods. Secondly, the utility module maintains parameters that need to be accessed by more than one module within a node. One example is the position of a node, which is calculated and updated by the mobility module, but also needed by the physical layer and potentially the localization module.

Modeling Framework

In this section, we discuss the basic modeling approaches, the assumptions behind these approaches, and implementation relevant aspects such as model abstraction level and model support for trading off accuracy and calculation complexity. The modeling framework is an adapted version of the OMNeT++ standard wireless communication framework, MiXiM [11], which is a complete framework designed for general wireless and mobile network simulations. We implemented a set of modules which are required for WMN simulations and defined a new base module.

Wireless channel models

The channel models express radio propagation effects as time variant factors of the instantaneous Signal-to-Noise Ratio (SNR) γ of the received signal. Although such SNR-based models abstract the exact signal behavior, e.g. the current phase shift, they enable the separate calculation of channel effects and, thus, adjusting the required accuracy by selecting the modeled effects and time-scale. On this SNR-level, we included the following widely accepted channel models for path-loss, shadowing, large and small-scale fading (adapted from MiXiM). Small-scale fading, i.e.

determining the variation of a wireless channel at small time scale, is caused by mobility in the propagation environment.

We model small-scale fading using the typical "Jakes-like" method [12] with the "land mobile" Autocorrelation Function (ACF) (Table 2.1 in [13]). This model is parameterized by a maximum Doppler shift according to carrier frequency *fc* and velocity v of the fastest moving object in the propagation environment, e.g. a moving user. This fading model is based on an Non-Line Of Sight (NLOS) assumption modeled by Rayleigh distributed signal amplitudes resulting in an exponentially distributed instantaneous SNR $\gamma_{i,j}$ for the channel from user i to user j. The model supports frequency-selective fading which appear in WMNs, which is parameterized by the mean delay spread and can be easily extended to further dimensions of the signal, e.g. spatial fading for multi antenna systems.

While the model as such supports reciprocal channels and correlation between multiple signal dimensions, in typical scenarios, the instantaneous SNR is assumed to be independent and identically distributed (i.i.d.) for different channels (i; j). Finally, the effect shadowing abstracts many physical effects such as reflection, diffraction, scattering, and absorption. Typically, shadowing is modeled by i.i.d. log-normal attenuation reflecting urban environments.

To implement shadowing and fading a block model has to be used requiring that $\gamma_{i,j}$ stays constant during a so-called block time. This interval typically refers to a Physical layer (PHY) inter leaver block or to the minimal coherence time of the channel. Therewith, each of these blocks experiences a quasi-static channel while the ACF defines whether consecutive blocks fade independently. Although the above models are widely-used, they are of course not suitable for any situation. For example, a Nakagami-type amplitude distribution may be preferred for scenarios with a LOS component, or a different ACF may be chosen to model an open-space scattering environment. Nonetheless, due to their clear separation, all these components can be exchanged easily and can be used to derive further model variants. A standalone implementation of these models for OMNeT++ is available in the ChSim module.

Physical layer models

The required physical layer models for mesh network simulation were adapted from the MiXiM framework [11]. At the physical layer, essentially the used modulation and Forward Error Correction (FEC) coding and decoding functions define the bit error rate and throughput of a system. As for the effects of wireless channels, the effect of these functions can be modeled at SNR-level. At this level, FEC introduces a so-called coding gain at the receiver, which can be expressed by a factor g to the SNR of the detected signal. This coding gain depends on the used code, its rate Rc, and the employed decoding algorithm. While an uncoded transmission is expressed by $g = 1$, typical channel codes provide coding gains larger than 2. Typically, the SNR threshold calibrates the system to stay below a given Packet Error Rate (PER) bound, e.g. as defined in the standard of the communication system [14]. It is selected a-priori depending on the receiver sensitivity for the chosen modulation scheme as given in the transceiver data sheet or approximated. By selecting thresholds and coding gain independently per terminal, terminals employing different PHY parameters can be modeled. Furthermore, by varying thresholds and coding gain over time, rate adaptation is supported.

In addition to systems detecting a bit from a single channel, this SNR-based model easily extends to diversity receivers where several channels are joined before the bit detection is made. Such systems exploit differently faded channels and employ a filter, e.g. Maximum Ratio Combining (MRC) [15], to combine the signals received from L channels to a single signal used for bit detection. For each employed channel, a different code or modulation may be chosen by defining an independent coding gain or threshold. This combining model can be used to model diversity

receivers combining signals in different dimensions, e.g. OFDM subcarriers, or multi-antenna systems. The physical layer modeling details are described in depth in [11].

Connection Models

Connectivity modeling is a challenging task in wireless simulations. In wired simulations, two nodes are connected by wires, which can be easily modeled (e.g. in OMNeT++ by connections). In wireless simulations, however, the "channel" between two nodes is the air, which is a broadcast medium and cannot be easily represented by one connection. In the mesh simulation framework, we decided to divide the modeling into two parts as described in *ChSim* in OMNeT++. The first part is the wireless channel and its attenuation property. The second part is the connectivity between nodes. Theoretically, a signal sent out by one node affects all other nodes in the simulation (if operating in the same frequency range). However, the signal is attenuated, so that the received power at nodes very far away from the sending node may be so low that it is negligible. In order to reduce the computational complexity in the framework, nodes are connected only when they are within the maximal interference distance. The maximal interference distance is a conservative bound on the maximal distance at which a node can still possibly disturb the communication of a neighbor. Please note that the maximal interference distance does not specify the maximal distance at which messages can be (correctly) received. A connection between mesh nodes is probably better defined by its complement: All nodes that are not connected definitely do not interfere with each other. Following this concept, a mesh router that wants to receive a message from a peer router, also receives all (interfering) signals and can, thus, decide on the interference level and resulting bit errors. The environmental model which defined in the mesh simulation framework is inspired by the MiXiM framework model, and allows more complex modeling of the simulation environment, especially when the WMN is deployed over a city-wide scale. In MiXiM, the environmental model provides the *ObjectManager* as a central authority for managing objects in the propagation environment. Objects are characterized by dimensions, position, angle of rotation (optionally), and frequency-dependent attenuation factors. An object that obstructs the line-of-sight between any pair of interconnected nodes causes additional signal losses during transmission as shown in Figure 3. Since entities can be mobile, intersections of the line-of-sight of two nodes with one or more objects must be determined at runtime. For any intersection with an object, its frequency-dependent attenuation factor is added up to yield the additional attenuation caused by the objects.

Utility Module

The utility module implemented in the mesh simulation framework has the following main objectives.

- Allows the experimenter to collect data, like throughput or delay.
- Reuse of the implemented code and compare it with other protocols under different/similar circumstances.
- performance instrumentation remains with the protocol code
- the instrumentation code is independent from the analysis tool used to gain insight

The utility module is a linear adaptation of the OMNeT++ base utility module. A detail on the generic utility module is available in [16]. The blackboard contained within the utility module provides a general solution for this problem. The instrumented module ("publisher") publishes the observed parameter on the blackboard. The meaning of a parameter is encoded in its class, as standard for object oriented languages. The blackboard then informs all parties that are interested in this particular parameter ("subscribers"). Often, the publishing module is the instrumented protocol, while the subscriber is usually a module that performs statistical analysis. This allows each researcher to

plug in his preferred statistical analysis method. At the same time, the overhead for the instrumentation (publishing) is a simple function call that has a constant execution time for each parameter. Of course, the execution time increases if more subscribers are interested in this particular parameter. The same concept is used for parameters, which need to be accessed or updated from multiple modules within a node. The publisher can publish the parameter as described above. Every subscriber will get a notification about the change and can take appropriate action. The utility module can also keep local copies of parameters published on the blackboard. This way, the utility module complements the push interaction of the blackboard with a pull interaction for appropriate parameters.

Protocol Library

We incorporated the already large protocol base on OMNeT++ to be adapted with the simulation framework through the base module. This lets the user of the simulation framework to replace, add or override modules to the base implementation. A biggest advantage of such an approach is to simulate wireless mesh networks irrespective of the underlying technology. For example, a user can perform IEEE 802.11-based WMN simulations in the first setting, and then replace the simulation modules to perform an IEEE 802.15.4-based WMN simulation using either the same/different settings.

MAC protocols

OMNeT++ also already implements standard MAC protocols for wireless Local Area Networks (LANs) and Personal Area Networks (PANs). The IEEE 802.11b/g family, as well as the IEEE 802.15.4 standard are ported from the Mobility Framework to the simulation framework. The base structure of the simulation framework also makes it very easy to implement new MAC protocols. For instance, the "FrameTimer" support modules can be used to easily implement Time Division Multiple Access (TDMA) based or other hybrid protocols.

Network layer protocols

The simulation framework supports networking protocols for a wide variety of traffic paradigms (source-to-sink; any-to-any; local neighborhood; etc), and these are further supported by the other simulation modules, e.g. localization data for geographic routing, motion data derived from the mobility module for decisions regarding when to update routes, or network-wide timers for synchronized protocols. We used the simulation framework to test the feasibility of the TBMGA protocol [17] and the results are discussed in the next section.

Mobility models

There is already a rich library of mobility modules implemented for OMNeT++, which includes simple modules like "constant speed mobility" and "circle mobility", but also modules that parse ANSim trace files and BonnMotion files. It is also very easy to create new mobility modules, by sub-classing from the *BaseMobility* class. The *BaseMobility* class provides all the functionality needed for mobility handling in the simulation framework – only the specific mobility pattern has to be implemented in order to create a new mobility module.

Implementation and Evaluation of TBMGA protocol in the Mesh Simulation Framework

In this section, we discuss the implementation and evaluation of the TBMGA routing protocol for WMNs in the mesh simulation framework. TBMGA is a proactive routing protocol which builds new routes from the gateways toward each node of the network during fixed intervals (e.g. every 10 sec.). During the time interval between each route discovery, nodes keep on exchanging probe packets for the purpose of estimating the ETX metric [18][19], which is used during route discovery. ETX is not sensitive to the load and this is necessary to avoid route flapping.

Being a proactive routing protocol, TBMGA deals with route table management. There are three types of tables: the Gateway Routing Table (GWRT), the Mesh Node Routing Table (MNRT) and the Cache Routing Table (CRT). TBMGA has been designed to allow the clients to connect to a wired network (in particular the Internet) but not to communicate directly among each others. Once the routes are established, the nodes must be aware of the distribution of the load and the current free capacity at the gateways to achieve good performance, load balancing and fairness. For this purpose, TBMGA is associated with a Gateway load balancing scheme, detailed in the following sub-sections.

Root Announcement (RANN), Route Reply (RREP), and Unicast packets are the message types defined by TBMGA. The tree topology formation begins when the root (Gateway) starts periodically (proactively) sending out the Root Announcement (RANN) message by increasing the sequence number with every announcement. The RANN are sent by the Gateways in broadcast and the dissemination covers the entire network. To avoid the uncontrolled spread of RANNs, each mesh node stops to rebroadcast the same RANN (same combination of Gateway Address and Sequence Number RANN) after a predefined time since it has received a new RANN even if it has a better metric. During this processing of the RANN message, each node learns the new route toward each gateway (each node receiving a RANN caches a route back to its originator). To make the gateways aware of the new routes, the nodes unicast an RREP message along the new paths to each gateway. The final result of this node discovery process is the creation of a tree from every gateway to every mesh node as depicted in Figure 3.

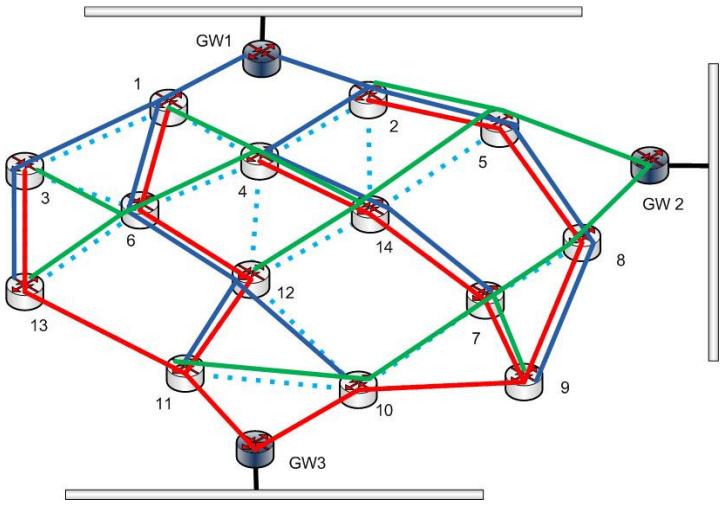

Figure 3: Three gateways building the trees throughout the network

Each Gateway broadcasts the RANN message in a proactive manner, where by at a fixed time, it is possible to build new update routes towards all the nodes of the networks according to the ETX metric. Any Mesh Node receiving a RANN behaves as represented in Figure 4.

- It refreshes the ETX metric of the RANN packet
- It checks the Gateway Address and the SN and looks up in its Route Cache Table
- If the combination is not still present in the Cache (new Gateway or known Gateway with new SN), it caches the mesh node it received the announcement from as the potential parent toward that Gateway, starts the Timer in the Route Cache Entry and rebroadcasts the RANN with the updated cumulative metric
- If the combination is already present in the Cache (same GW Address and SN) and the Timer field is not still elapsed, it checks if the ETX metric of the new RANN is better than the one already stored. If it is better it updates the Cache with the new information and rebroadcasts the RANN with the updated cumulative metric. If the ETX metric is worse, it discards the RANN
- If the combination is already present in the Cache (same GW Address and SN) but the Timer is already elapsed, it discards the RANN

When the Timer in a Route Cache Entry ends, the Mesh Node updates the corresponding Gateway Routing Table (GWRT) Entry with the new metric and next hop to the Gateway. In this way, at the end of the RANN's dissemination all over the network, each node learns the best route to the Gateway which originated the RANN.

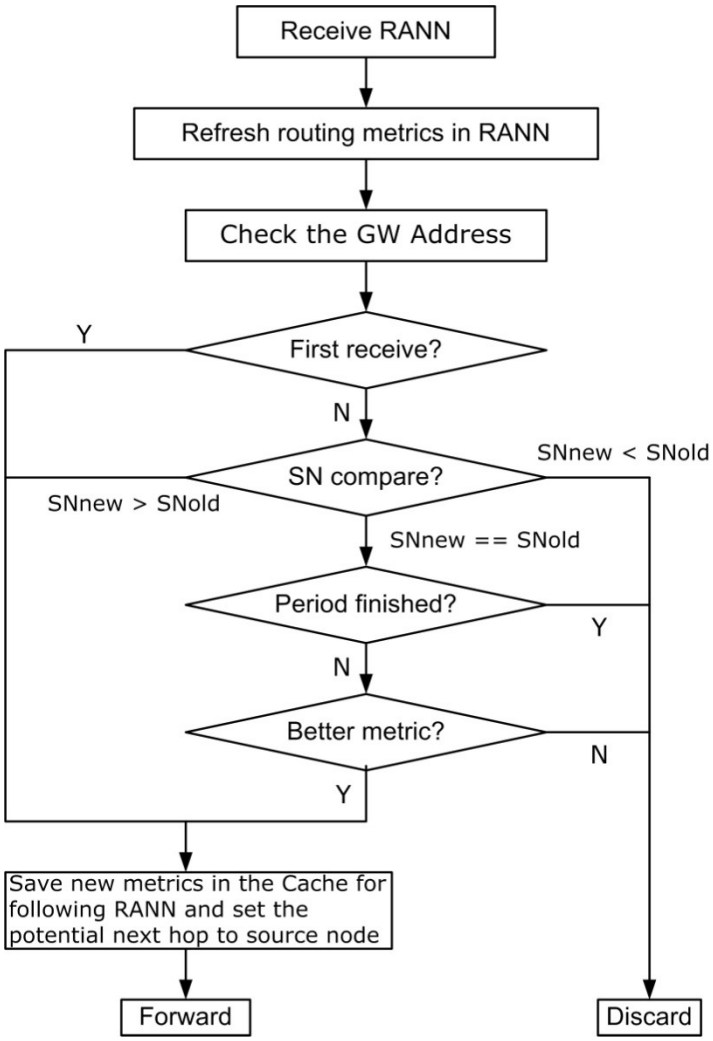

Figure 4: the process followed by a mesh node upon receiving a RANN is schematized in the figure.

The procedure described above permits the Mesh Nodes to know the best route toward a Gateway. Anyway, to guarantee a two-way communication, the Gateway must be aware of the new routes to reach the Mesh Nodes. To perform this task, each Mesh Node, after having chosen its parent node toward a Gateway, unicast an RREP along the new path to the Gateway:

- When the Timer in a Route Cache Entry ends, the Mesh Node updates the corresponding Gateway Routing Table (GWRT) Entry with the new metric and next hop's MAC Address to the Gateway
- At the same time the Mesh Node sends to its next hop toward the Gateway a Route Reply (RREP) message with the Gateway's address as the destination address
- Each intermediate mesh node that receives the RREP forwards the message to its selected parent node toward the Gateway contained in the RREP and updates the corresponding Mesh Node Routing Table (MNRT) Entry with the next hop's MAC Address toward the MN originating the RREP

- When the RREP arrives to the root, the Gateway updates its Routing Table Entry for the Mesh Node which has sent the RREP with the next hop MAC Address towards that node

Note that an RREP is never sent by a node on behalf of other mesh nodes like it can happen in MANET routing protocols like AODV. This ensures that the discovered path metric is current, since the route request and the route reply traverse the complete path from the gateway to each node and collect the current metric values.

Once the next-hops from/to the Gateways are computed, the data packets can be routed toward the destination through Unicast packets. The source node will encapsulate the Ethernet payload in a Unicast packet. The intermediate nodes will simply forward the Unicast packet according to its destination MAC address.

Gateway Load Balancing

As stated in [21], the availability of multiple Gateways does not imply a direct and linear increase of network performance. The presence of more Gateways can be reflected as a network capacity benefit only with the adoption of a suitable load balancing technique. This is mainly due to the uneven load distribution experienced across a real network caused by the not uniform geographical dislocation of end-users and by the differences in the traffic generated by different clients.

Gateway Load Balancing provides a natural and elegant way to spread the traffic over different paths toward a set of gateways [22]. With TBMGA, since every MN conserves the next-hop toward each Gateway in its GWRT table, this process is particularly straight-forward. The proposed gateway load balancing scheme is tailored to suit our needs from the LDAB algorithm, presented in [20]. LDAB tries to reach load balancing and fairness among the nodes monitoring the queue length at the gateways over a fixed time period. If the average queues length gets over a certain threshold value, the GW indicates to its active source (sources with high traffic) to connect to other GWs if possible. The choice to redirect only the active source and not all the flows connected to the GW is very important to avoid route flapping, which is a common problem of many load balancing approaches.

We modified this approach to adapt the way in which the active sources are chosen. If we simply base our choice according to the volume of the traffic generated by the MNs we could switch a MN that is only one hop far away from its current GW, and as such it should reasonably stay connected to it. We thought it would be more sensible and advantageous to take into account both the traffic rate, the loss ratio and the number of hops to the GW. Two MNs, transmitting approximately with the same rate, will be given different weights based on their distance to the GW and the loss ratio along the path. In this way, we increase the likelihood to keep the MNs in proximity of a GW connected to it while switching the ones far away but most probably closer to others GWs.

To accomplish this we consider as "the active sources" the MNs with the highest value *[(MN's ETX metric)*(MN's rate)]* where the ETX metric allows to consider both the number of hops and the packet loss.

The pseudo-code in Figure 5 summarizes the main steps carried out by the load-balancing algorithm. Each GW keeps checking its queue length. In case the average queue length exceeds a fixed threshold during a time window, the GW identifies the one with the highest value among the MNs currently connected to it. At this point, it sends a CHANGE_Pkt request to this MN asking to switch to another GW if possible.

```
At a GW:
sending
If (the average queue length for a time period (Monitor_Cycle) > Threshold)
  Identify the MN sending to me with the worst (ETX metric*rate)
  Send a CHANGE_Pkt message to switch GW, if possible
End if
receiving
If a GW_REQ arrives from a MN:
  If (the average queue length < Threshold)
    Admit this node sending a GW_REP to it
  End if
End if

At a MN:
sending
When a CHANGE_Pkt arrives from the default GW:
  For (each GW in the GWRT != default_GW)
    Send in sequence, according to the best ETX metric,
    a GW_REQ with the MN's estimated traffic
  End for
receiving
The first GW replying with a GW_REP to a GW_REQ becomes the new default_GW
```

Figure 5. Pseudo code for the gateway load balancing algorithm

When receiving a CHANGE_Pkt, the MN looks up an alternative gateway in its table. If other Gateways are available, the MN sends, in sequence, according to the best ETX metric, a Gateway Request (GW_REQ) message with the MN's estimated traffic to all of the other gateways. The Gateways receiving the GW_REQ check their availability to admit the new MN against the MN's estimated traffic. If a GW can afford the new MN, it replies with a Gateway Reply (GW_ REP). The MN will associate with the first GW replying. After the reception of the GW_REP, the MN redirects all of its further traffic flows toward its new Gateway.

If however, a MN is not accepted by any other GW, it keeps on sending its flows to the current default GW. In such a case, the GW will send the CHANGE_Pkt to another MN serviced by it, always according the {(MN's ETX metric)*(MN's rate)} criterium.

Simulation Evaluation

In this section, we describe the simulation evaluation of the TBMGA routing protocol using the mesh simulation framework. We ran the simulation for 300 sec. We used IEEE 802.11g as the underlying MAC protocol. The backhaul is always composed by 13 fixed MNs, while the number of Gateway varies from 1 to 2 as depicted in Figure 6. The traffic is generated by three clients who transmit at different rate. The packet size was set to 512 bytes.

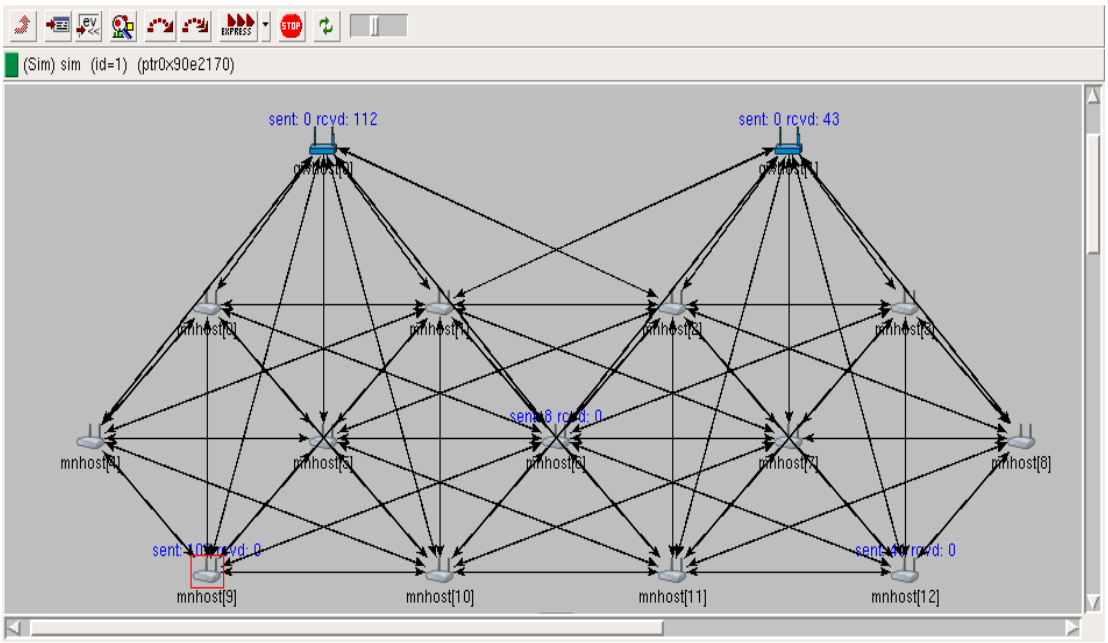

Figure 6. Scenario used in simulations.

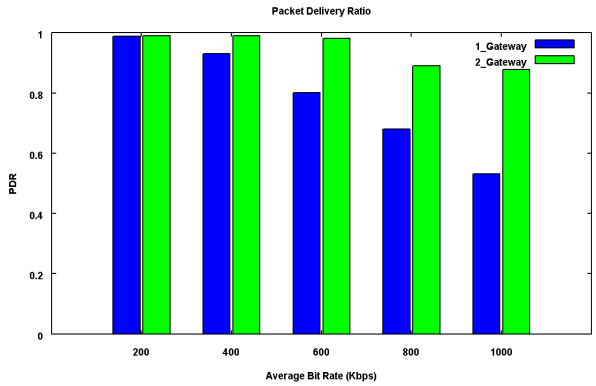

Figure 7. Packet Delivery Ratio at different rate and number of Gateways.

In Figure 7 we compare the Packet Delivery Ratio at different rates and using a variable number of Gateways ranging from 1 to 2. The x-axis represents the average rate of the three flow, e.g. for 800Kbps we use flow1:600Kbps, flow2:800Kbps, flow3:1000Kbps. When using 1 Gateway the PDR decreases quite rapidly as the rate of the flows increases. This is due to all the three flows being connected to the same gateway which leads to heavily congested paths particularly in proximity of the Gateway. The ETX metric which avoids routes with high loss rate is not sufficient to mitigate the lack of proper resources usage. With the introduction of more Gateways we can exploit the Gateway Load Balancing feature to reduce the congestion along the routes. We can observe the PDR improving at every rate once we introduce 2 gateways.

As expected, we can observe the same trend for the end-to-end delay. Figure 8 compares the average end-to-end delay of packets for different bit-rates and with the number of Gateway ranging from 1 to 2. The introduction of more Gateways implies the traffic to be split along different paths, so that we get a better spatial usage of wireless bandwidth and consequently a decrease in the number of congested routes. As an obvious result the average end-to-end delay decreases.

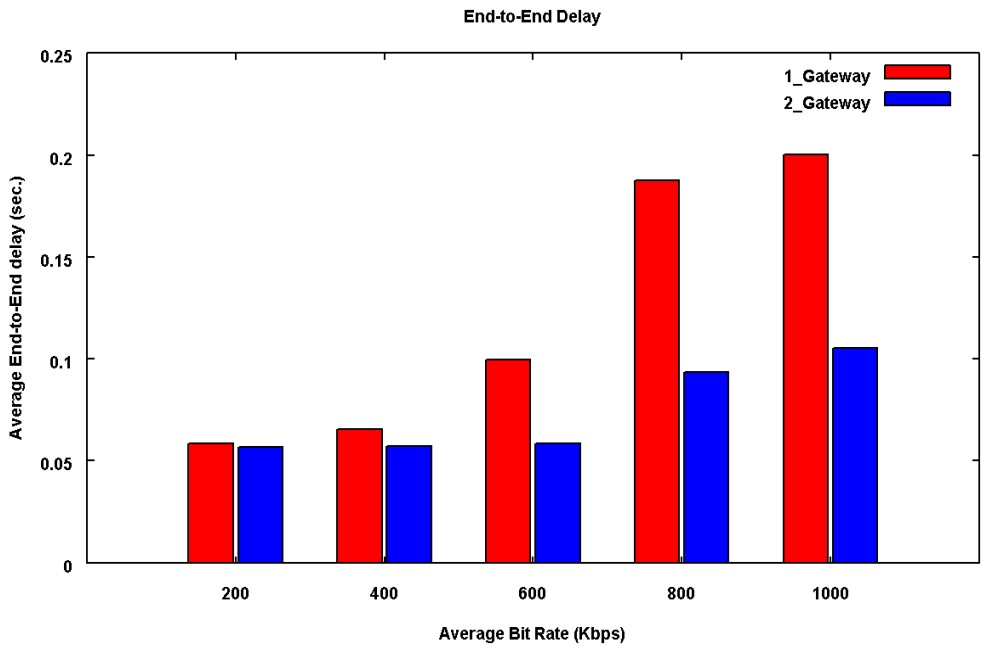

Figure 8. Average End-to-End delay at different rate and number of Gateways.

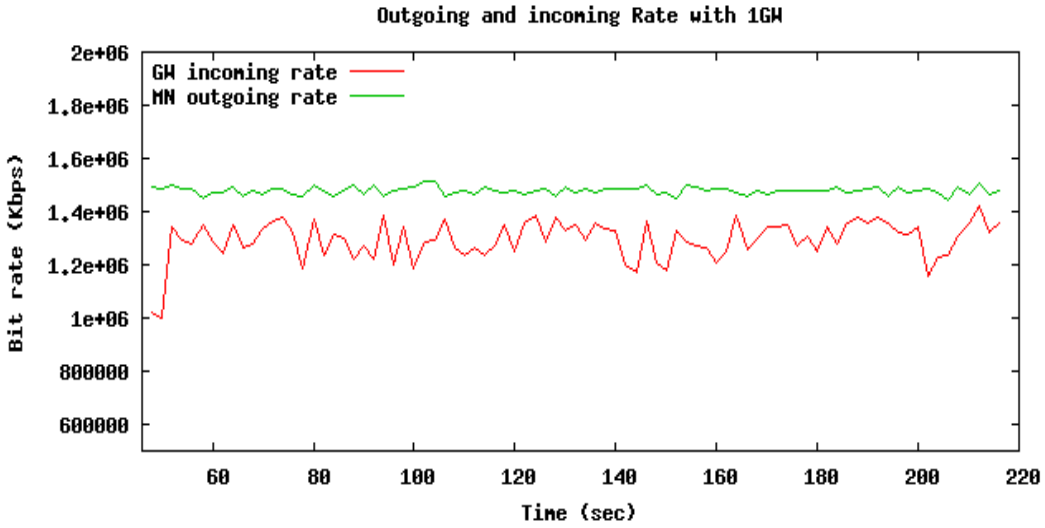

Figure 9. Instantaneous rate generated by the MNs and received at the GW.

The following two figures are a comparison between the instantaneous incoming rate that we get at the gateways with the instantaneous outgoing rate generated by the mesh nodes. Clients MN9, MN6 and MN12 generate traffic flows f1, f2 and f3 at the rate of 720, 240 and 480 Kbps respectively. We start the flows after 46 sec. the beginning of the simulation. We consider the sum of the incoming rate at the GWs and the sum of the outgoing rate at the MNs. In the ideal situation this two parameters should be very close. Nevertheless, since we are coping with multi-hop wireless networks, there could be a lot of packets dropped due to potential buffer overflows. The problem is not only due to the high concentration of traffic at a gateway, that leads to saturation, but more generally it can be observed all over the path from a MN to a GW due to the shared nature of the wireless medium. The presence of multi-gateway is hence important not only to alleviate the burden at the gateway, but also to spatially differentiate the routes followed by the flows en route to the gateways.

The scenario used for the simulations is always the same (cf. Figure 6). The first test was carried out using only one GW1. As expected, as in figure 8, in this situation, the multi-hop backbone is not able to deliver all the packets generated by the MNs, due to saturation and congestion along the common links and at the GW which leads to MAC buffer overflow and thus packet drops. The red curve, representing the instantaneous incoming rate at the GW, is constantly and abundantly under the green one, which represents the sum of the instantaneous outgoing rates of the MNs.

In the second test we used two GWs. The presence of the second gateway allows the use of the multi-gateway association and of the load balancing algorithm. The situation is highly improved as shown by the almost overlapping of the two curves (figure 10). The incoming rate at the GWs and the outgoing at the MNs being almost the same imply that the numbers of packets dropped along the paths to the GWs are diminished. The presence of multiple gateways permits to balance the traffic load over different GWs, and also possibly along the routes followed by the packets on their way to the GW, with the obvious consequence to reduce the per-node contention period for accessing the wireless medium, and thus reduce the number of MAC buffer overflows.

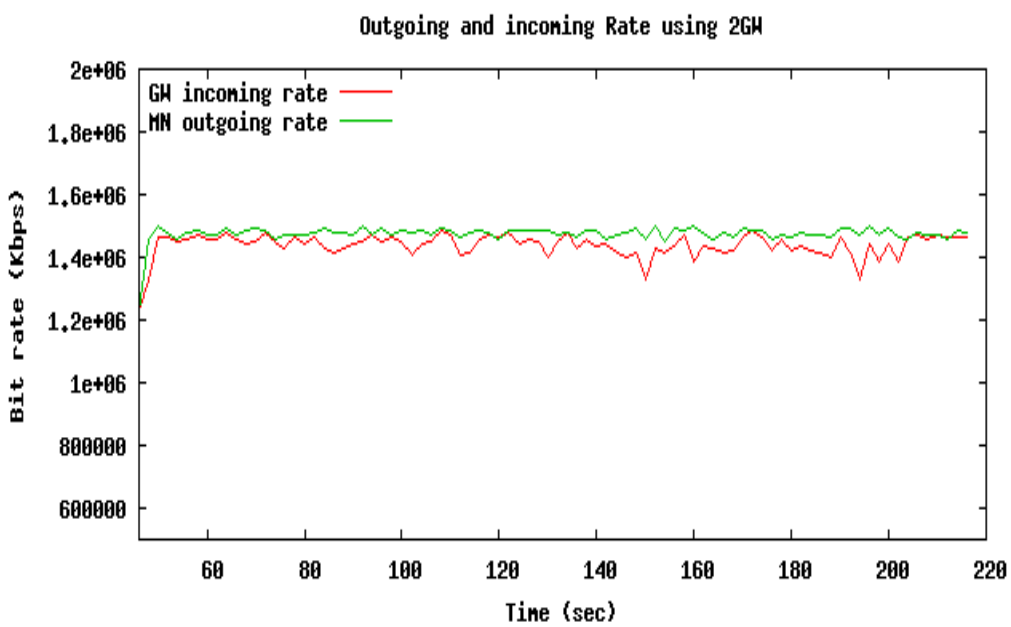

Figure 10. Instantaneous rate generated by the MNs and received at the GWs.

Conclusions

In this report, we introduced a new scalable wireless mesh network simulation framework which provides a powerful simulation framework and concise modeling chain for Wireless Mesh Networks. The mesh simulation framework is still in development, even though it already provides a solid base of available models and implementations integrated into the simulation framework environment, including models for mobile environments, nodes, radio propagation models for multiple signal dimensions, physical layer models for modulation, coding and diversity receivers as well as an extensive library of MAC and network protocols. The major activity of the simulation framework development was in the integtration of the scattered models in the OMNeT++ simulator into a common framework for simulation of WMNs. Thus, the simulation framework profits from the rich experience gathered from the wireless simulation frameworks of OMNeT++. A modular design enables the straightforward definition of complex scenarios as well as the easy integration of new models and protocol implementations.

It is, therefore, our hope that the mesh network simulation framework can provide a clear structure and its extensive modeling base and the ongoing development of the integrated simulation-emulation environment can motivate researchers to contribute to the maturity of the mesh network domain from a simulation perspective. The emulation framework development, which is currently in progress will enable the WMN research community and the practitioners alike to simulate and parallel run real WMN networks, and to validate the performance, thus bridging the simulation-real network performance gaps.

References

[1] J. C. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In 11th Annual International Conference on Mobile Computing and Networking (MOBICOM 2005), pages 31–42, Cologne, Germany, August 28 - September 2 2005.

[2] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In 10th annual international conference on Mobile computing and networking (MobiCom '04), pages 114–128, New York, NY, USA, 2004.

[3] R. Karrer, A. Sabharwal, and E. Knightly. Enabling large-scale wireless broadband: The case for taps. In 2nd Workshop on Hot Topics in Networks (Hot-Nets II), Cambridge, MA, USA, November 2003.

[4] S. Ivanov, A. Herms, and G. Lukas. Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. In 4th Workshop on Mobile Ad-Hoc Networks (WMAN'07), pages 433–444, Bern, Switzerland, February 26 - March 2 2007.

[5] M. Engel, M. Smith, S. Hanemann, and B. Freisleben. Wireless ad-hoc network emulation using microkernel-based virtual linuxsystems. In 5thEUROSIM Congress on Modeling and Simulation, pages 198–203, Cite Descartes, Marne la Vallee, France, September 6-10 2004.

[6] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In 3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '02), pages 104–111, New York, NY, USA, 2002. ACM.

[7] T. Krop, M. Bredel, M. Hollick, and R. Steinmetz. Jist/mobnet: combined simulation, emulation, and real-world testbed for ad hoc networks. In WinTECH '07, pages 27–34, New York, NY, USA, 2007. ACM.

[8] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In IEEE Wireless Communications and Networking Conference (WCNC 2005), volume 3, pages 1664 – 1669, March 2005.

[9] A. Zimmermann, M. Gunes, M. Wenig, U. Meis, and J. Ritzerfeld. How to study wireless mesh networks: A hybrid testbed approach. Advanced Information Networking and Applications, 2007 (AINA '07), pages 853–860, May 2007.

[10] E. Weingartner, F. Schmidt, T. Heer, and K. Wehrle. Synchronized network emulation: matching prototypes with complex simulations. SIGMETRICS Perform. Eval. Rev., 36(2):58–63, 2008.

[11] S. Valentin et. al. Simulating Wireless and Mobile Networks in OMNet++: The MiXim Vision.In proceedings of OMNeT++ workshop (SIMUTOOLS), Marseille, France, 2008.

[12] J. Cavers. Mobile Channel Characteristics. Kluwer Academic, 2000.

[13] M. K. Simon and M.-S. Alouini. Digital Communications over Fading Channels. John Wiley & Sons, Inc., 2 edition, 2004.

[14] B. O'Hara and A. Petrick. IEEE 802.11 Handbook: A designers companion. IEEE Press, 1999.

[15] J. G. Proakis. Digital Communications. McGraw-Hill, 4 edition, 2000.

[16] Mobility framework (MF) for simulating wireless and mobile networks using OMNeT++. [online]. Available: http://mobility-fw.sourceforge.net/.

[17] T. Rasheed and S. Maurina, On Tree-Based Routing in Multi-Gateway Association based Wireless Mesh Networks , submitted to 20th International PIMRC Conference, 2009.

[18] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing", in ACM Mobicom, 2003.

[19] R. Draves, J. Padhye, and B. Zill, "Comparison of routing metrics for static multi-hop wireless networks", in ACM SIGCOMM, 2004.

[20] D. Nandiraju, L. Santhanam, N. Nandiraju, and D. P. Agrawal, "Achieving Load Balancing in Wireless Mesh Networks Through Multiple Gateways", *IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pp. 807-812, Vancouver, 2006.

[21] S. Lakshmanan, K. Sundaresan, and R. Sivakumar, "On Multi-Gateway Association in Wireless Mesh Networks", *in 2nd IEEE Workshop on Wireless Mesh Networks*, WiMesh 2006, pp.64-73, Reston, VA, USA.

[22] K Ramachandran, M. M. Buddhikot, G. C. Menon, S. Miller, K. Almeroth, and E. B-Royer, "On the Design and Implementation of Infrastructure Mesh Networks", WiMesh, Santa Clara, 2005.