A new approach to simulating PHY, MAC and Routing

Nicola Baldo
University of Padova
via Gradenigo 6/B, 35131
Padova, Italy
baldo@dei.unipd.it

Federico Maguolo
University of Padova
via Gradenigo 6/B, 35131
Padova, Italy
maguolof@dei.unipd.it

Marco Miozzo
Centre Tecnologic de
Telecòmunicacions de
Catalunya (CTTC)
Av Canal Olímpic s/n, 08860
Castelldefels, Barcelona
(Spain)
marco.miozzo@cttc.es

ABSTRACT

In recent years, network simulation has become a very difficult task due to the proliferation and integration of wireless technologies. In this paper, we discuss the new challenges that have arisen regarding the simulation of the wireless channel and the PHY, MAC and Routing layers, argumenting why currently available network simulation tools such as ns2 and many of its recently proposed extensions do not address all these issues in a comprehensive and systematic fashion. We then present a novel framework designed to address these challenges. This framework has been developed as an extension of NS-Miracle, in order to have support in the definition and management of scenarios involving the use of multiple interfaces and radio technologies, and is made up of two components. The first component is the Miracle PHY and MAC framework, which provides support for the development of Channel, PHY and MAC modules, providing support for features currently lacking in most state-of-the-art simulators, while at the same time giving a strong emphasis on code modularity, interoperability and reusability. The second component is the Miracle Routing framework, which enables the integration of different routing schemes in a multi-tier architecture to provide support for the simulation of multi-technology and heterogeneous networks. We want to observe that, thanks to this framework, it is now possible to carefully simulate complex network architectures potentially at all the OSI layers, from the physical reception model to standard applications and system management schemes. This allows to have both a comprehensive view of all the networks interactions and its high level view, which plays an important role in many research investigation area, such as cognitive networking and cross-layer design.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: General, Model Validation and Analysis, Model Development; C.2.6 [**Computer-Communications Networks**]: Internetworking

General Terms

PHY, MAC, Routing, Multi-technology, Cross-layer, Heterogeneous Networks

1. INTRODUCTION

The recent progress in communications technology has seen a remarkable increase of the number and types of wireless interfaces being bundled with mobile devices. In turn, research in wireless communications has faced several challenges in studying each of these technologies, as well as in investigating suitable strategies to make them coexist and interoperate. As a consequence of these facts, there has been an increasing need for investigation tools, in particular network simulators, to be conveniently exploited for these research purposes.

Many are the features that researchers seek in network simulators. In this paper, we focus on the issues which in our opinion are not addressed to a good extent by the current network simulators. These issues are:

- **Accurate channel and PHY layer modeling**: recently, there has been an increasing awareness of the need for accurate modeling of channel and PHY layer aspects. While simplified models, such as the disc propagation model, are still useful in some contexts, a general purpose simulator is nowadays expected to provide more realistic simulation of the signal propagation and reception processes.

- **Modeling of a complete system**: the increasing complexity of communication systems has made performance evaluation a really complex task, due to the fact that often subtle interactions among the different components of the system play an important role in determining the overall performance. These interactions are often not evident when only one or a few of the components of a communication system are modeled. This need is become crucial in many novel network research areas, such as Cross-layer design [1, 2] and Cognitive Networking [3–5]. For this reason, a good general purpose network simulator is nowadays expected to provide means of modeling a complete communication system, from channel and PHY layer modeling all the way up through the protocol stack to the application layer.

- **Rapid prototyping of new wireless technologies**: new wireless technologies have been proposed and released at an amazing pace in recent years, and it is commonly the case that researchers are struggling to develop simulation tools in a timely fashion to study these technologies as they emerge. In the past, the majority of the code taking care of wireless technology simulation (especially at the PHY and MAC layer) has always been designed for a specific technology, without addressing code reusability during the design phase of the code. The ns2 simulator, in particular, was not originally designed to simulate wireless networks, and support for this has been added only at a later step, and in a rather quick-and-dirty way. As a consequence, the task of properly designing and developing code for the simulation of new wireless technologies is very time consuming. Though wireless technologies can overall differ significantly from each other, it is to be noted that there are many aspects and components of the PHY and MAC layers which are very similar among different technologies. An ideal simulator would leverage on well-defined APIs for PHY and MAC modules to allow easier and faster development of simulator code, with a particular emphasis on modularity and reusability.

- **Spectrum Awareness**: recently, research in fields such as Cognitive Radio, Ultra Wide Band and Underwater communications systems have created a strong need for network simulators to provide a better modeling of the Radio Frequency aspects of communications systems, such as spectrum occupation, RF filtering and inter-channel interference. State-of-the-art network simulators such as ns2 lack support for a correct modeling of how communication systems make use of the frequency spectrum.

- **Inter-technology interference**: unlicensed bands, in particular the 2.4 GHz ISMband, are characterized by the simultaneous presence of different wireless technologies interfering with each other; this is the case of popular wireless communication technologies such as 802.11, Bluetooth and WiMAX, as well as non-communicating technologies such as microwave ovens. Therefore, the introduction of spectrum awareness in wireless network simulators should be made in such a way to allow proper accounting of how different technologies interact among themselves in the propagation medium. We note that this goes beyond the issues discussed above. In fact, support for spectrum awareness in a single-technology scenario can be introduced by performing custom modifications to the code implementing that particular technology, while the simulation of inter-technology-interference requires that all technologies use the same representation of interference and spectrum usage.

- **Multi-technology multi-interface communication capabilities**: as more and more devices nowadays are equipped with multiple interfaces using different communication technologies, network simulators should provide support for proper modeling of this type of scenarios, by means of a flexible and modular protocol stack architecture together with proper support for the development of the control modules which need to manage such a complex architecture. A key role with this respect is played by routing modules, which need to provide proper routing functionalities to enable the use of the multiple available interfaces.

- **Support for Heterogeneous Networks**: while traditional networking research focused mainly on homogeneous networks, such as infrastructured, ad-hoc, and mesh networks, in recent years there has been an increasing interest in scenarios in which these types of networks coexist. Simulating this type of scenarios today is very challenging, in particular due to the fact that the routing layer in state-of-the-art simulator is mainly designed for homogeneous networks. As a consequence, there is a need for supporting this type of heterogeneous network composition at the routing layer.

In this paper, we propose a framework for the simulation of the physical, medium access and network layer, which has the objective of addressing the above mentioned issues. This solution was developed on top of the NS-Miracle framework [6], in order to exploit its support for the coexistence of multiple interfaces and multiple radio technologies within the same node, which is not available in the original ns2. We also note that we evaluated the possibility of implementing the same solution on top of ns3, but we chose not to do it, since, at the time we took this decision, the ns3 code was still relatively immature, and most importantly it lacked wireless technology implementations other than 802.11. Still, we hope that the discussion we provide in this paper will also provide useful for the future ns3 development.

The framework we present in this paper consists of two main components:

- the Miracle PHY and MAC framework, which provides functionality for channel modeling as well as two APIs for easy development of respectively PHY and MAC layer modules, with a particular focus on code reusability across different wireless technology implementations;

- the Miracle Routing framework, which introduces a new paradigm in routing simulation, by means of which interfaces with different technologies and routing protocol implementations can be used together in multiple tiers, providing enhanced support for the simulation of 4G networks.

In the rest of this paper, we will provide a detailed description of these components. A detailed discussion of the characteristics of NS-Miracle is out of the scope of this paper; the interested reader is referred to [6].

2. MIRACLE PHY AND MAC

2.1 Related Work

Accurate Channel and PHY layer modeling has drawn a significant amount of attention in recent years, and much work has been done with this respect. The ns2 simulator, in particular, was well-known for its poor channel and PHY layer modeling, and for this reasons several enhancements have been proposed [7–10]. However, the addition of this functionality has always been done in a technology-specific manner, and often in a quick-and-dirt fashion. The ns2 mobile node, in particular, does not have a good separation of functionalities between the MAC and the physical layer; some key PHY functionalities, such as in particular the determination of the end of the reception of the packet, are performed at the MAC layer, making it difficult to introduce functionality such as interference calculation, as well as making the code notoriously hard to read and to debug. In fact, the introduction of features such as enhanced error and interference models has required significant modifications, such as what introduced in dei80211mr [8], and improving the architecture of the code for the sake of easier debugging and readability has required a complete redesign of the code, as is the case of the new 802.11 model in [9]. Moreover, introduction of support for new wireless technologies has required extensive tweaking, if not even the use of the ns2 code as a mere entry point for completely customized code, as is the case of [11]. To summarize, everytime a new technology is to be implemented, significant developer effort is needed, which is exacerbated by the fact that the basic ns2 mobile node does not provide good channel modeling, and that all extensions with this respect have been too much technology-specific and cannot be easily reused for different technologies. Regarding other simulators, it is to be noted that some of them, such as ns3, allow the specification of customized callbacks between modules at the different layers of the protocol stack. However, the type of interactions in use commonly varies depending on the technology being considered as well as on the particular implementation, and a well-defined set of callbacks to be exploited for channel, PHY and MAC layer modeling is still lacking. Finally, to the best of our knowledge, no well-known simulator provides a good and generic model for how different wireless technologies make use of the frequency spectrum and interact among themselves.

2.2 The Miracle PHY and MAC framework

The Miracle PHY and MAC framework was explicitly designed to overcome these issues. In our effort to develop a modular and extensible framework for Channel and PHY layer modeling, we chose an object-oriented design and we defined a set of classes, depicted in Figure 1.

The channel model we developed is based on the modeling of a fundamental entity, the PHY Layer Transmission (PLT) of a packet. We define a PLT by the attributes characterizing it; choosing these attributes corresponds to choosing our PHY and channel modeling assumptions. PLTs instances have a 1:1 association with ns2 Packet instances, and therefore PLT attributes are conveniently gathered in a new ns2 packet header named `MPHY_HDR`. The attributes we define for PLTs, and the consequent channel modeling assumptions we make, are the following:

- `duration`: a PLT is an event which extends over a given time interval. The length of this interval is given by the `duration` attribute.
- `Pt`: a PLT is characterized by its transmission power, as set by the PHY layer of the transmitter. This attribute refers to the PLT as a whole; in other words, power is considered to be constant during the whole duration of a transmission. This choice is intended to achieve a reasonable tradeoff between modeling accuracy and complexity. We note that, while this is the approach in use by the vast majority of simulators, it is nevertheless a simplifying assumption of which we should be aware.
- `Pr`: the process of receiving a PLT is modeled by mathematical operations performed on the `Pt` attribute, resulting in the `Pr` attribute which represents the received power. Examples of these interactions are propagation, antenna gains, RF filtering, and signal processing at the PHY layer.
- `Position`: a PLT is characterized by the position of the transmitter (`srcPosition`) and of the receiver (`dstPosition`). These attributes are references to instances of classes belonging to the Position class hierarchy as defined in NS-Miracle [6]. Therefore, they can provide additional information such as node mobility information. This can be used for enhanced channel modeling features, such as the determination of the effects of fast fading as a function of speed.
- `modulationType`: a value univocally identifying the particular modulation and coding scheme in use by the transmission being considered. The main purpose of this attribute is to provide support for modeling the acquisition process[1] in multi-technology scenarios. We note that this solution can accommodate perfect acquisition models (in which the receiver always acquires correctly signals of the desired type and discards all others) as well as more complex solutions such as stochastic models for preamble detection.
- `srcSpectralMask`: a transmission consists of power radiated non-uniformly into the spectrum. We assume that the specification of the power spectral density function of the transmission (normalized to the power associated with the transmission) can account sufficiently in detail for this issue. `srcSpectralMask` is a pointer to a class of the `MSpectralMask` hierarchy which is intended to implement this type of function. We chose to use an abstract class for this purpose, in order not to pose any particular limitation on how this function is implemented. Nevertheless, piece-wise constant or linear functions should be enough for most purposes, and for this reason the only implementations we provide for `MSpectralMask` use a rectangular function.
- `dstSpectralMask`: each reception process is characterized by a RF filter, which is represented by means of its frequency response. The implementation issues for this are the same as for the spectrum usage information discussed above; for this reasons, as we did for the power spectral density of PLTs, RF filters are represented by instances of the MSpectralMask class. The RF filtering process is modeled as a gain applied to the received power of the PLT; this gain is determined as a function of the spectral mask of the transmission and the frequency.
- `Pn`: the noise power at the receiver.
- `Pi`: the phenomenon of interference is summarized by the interference power attribute `Pi`, calculated by means of aggregation of all simultaneous PLT, and with respect to a particular PLT for which reception is being attempted. We do not pose any particular constraint on the exact model to be used for this purpose; rather, interference models are to be implemented by inheriting from the `MInterference`, and implementing the `addToInterference()` and `getInterferencePower()` methods according to the chosen model. We provide one particular implementation of this type of classes, named `InterferenceMIV`, which aggregates all simultaneous PLTs into a single piece-wise constant function of time using the well-known gaussian model (i.e., summing power values), and returns the total interference power on a given packet calculated as the mean integral value of the aggregation of interfering PLTs.

The other important element of our Channel and PHY modeling framework is the MPhy class: it is an abstract class which provides both channel modeling functionality and the API for the development of PHY layer implementations. Channel modeling is implemented by associating each MPhy instance with instances of other objects which implement the different components of a channel model. We define the following classes of objects for channel modeling:

- `MPropagation`: similarly to the Propagation class in ns2, implementations of this class will account for the attenuation

[1]by *acquisition process* here we mean the set of tasks such as preamble detection which are commonly performed by the PHY layer at the beginning of a reception

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2008.4441
http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4441

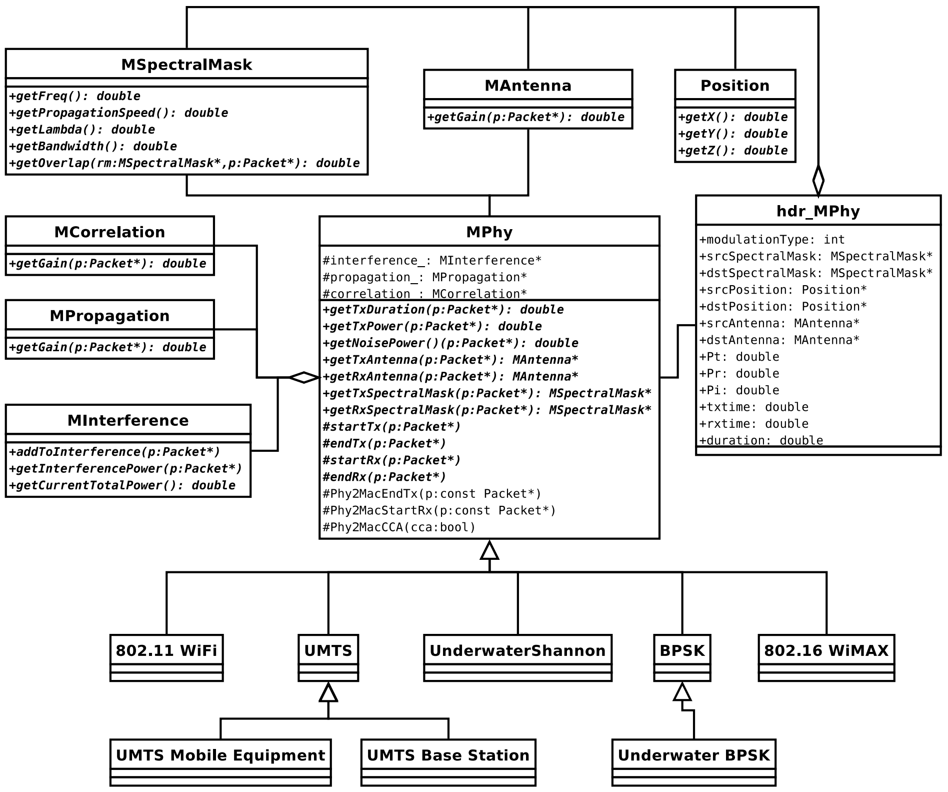
MSpectralMask
+getFreq(): double
+getPropagationSpeed(): double
+getLambda(): double
+getBandwidth(): double
+getOverlap(rm:MSpectralMask*,p:Packet*): double
MAntenna
+getGain(p:Packet*): double
Position
+getX(): double
+getY(): double
+getZ(): double
hdr_MPhy
+modulationType: int
+srcSpectralMask: MSpectralMask*
+dstSpectralMask: MSpectralMask*
+srcPosition: Position*
+dstPosition: Position*
+srcAntenna: MAntenna*
+dstAntenna: MAntenna*
+Pt: double
+Pr: double
+Pi: double
+txtime: double
+rxtime: double
+duration: double
MCorrelation
+getGain(p:Packet*): double
MPropagation
+getGain(p:Packet*): double
MInterference
+addToInterference(p:Packet*)
+getInterferencePower(p:Packet*)
+getCurrentTotalPower(): double
MPhy
#interference_: MInterference*
#propagation_: MPropagation*
#correlation_: MCorrelation*
+getTxDuration(p:Packet*): double
+getTxPower(p:Packet*): double
+getNoisePower()(p:Packet*): double
+getTxAntenna(p:Packet*): MAntenna*
+getRxAntenna(p:Packet*): MAntenna*
+getTxSpectralMask(p:Packet*): MSpectralMask*
+getRxSpectralMask(p:Packet*): MSpectralMask*
#startTx(p:Packet*)
#endTx(p:Packet*)
#startRx(p:Packet*)
#endRx(p:Packet*)
#Phy2MacEndTx(p:const Packet*)
#Phy2MacStartRx(p:const Packet*)
#Phy2MacCCA(cca:bool)
802.11 WiFi
UMTS
UnderwaterShannon
BPSK
802.16 WiMAX
UMTS Mobile Equipment
UMTS Base Station
Underwater BPSK

Figure 1: Class diagram for MPhy and related classes

of the power of a PLT due to effects such as path loss, fading, and shadowing;

- `MAntenna`: similarly to the Antenna class in ns2, this class hierarchy provides means of implementing the gain of directional antennas as a function of PLT attributes (the most useful for this purpose being the position attributes);
- `MCorrelation`: this class of objects is intended to implement the gains due to the signal processing performed by the receiver. A remarkable use case for this class of objects is the processing gain in DSSS and CDMA systems.
- `MInterference`: this class of objects is meant to provide the implementation for interference calculation.

The first three of the above mentioned classes are required only to implement a `getGain(Packet* p)` method which is expected to provide the gain value to be applied to a given PLT. On the other hand, the `MInterference` class, is required to produce the interference perceived by a particular PLT; this task is more complex since in general interference depends on all PLTs overlapping in time and frequency with the particular PLT being considered. For this reason, classes implementing `MInterference` are expected to keep track of all currently active PLT, by providing implementation of two methods: `addToInterference(Packet* p)`, which has to be called at the beginning of every PLT so it can be added to the set of active PLTs, and `getInterferencePower( Packet* p)`, which is to return the interference of all active PLTs on the given PLT.

The MPhy class is meant to provide support only for functionality which is shared by different channel models and wireless technology implementations. For this purpose, technology-specific PHY layer functionality is meant to be introduced by inheriting from the MPhy class and implementing the following virtual methods:

- `getTxDuration(Packet* p)`: must be provided by the transmitting PHY to determine the duration of a transmission.
- `getTxPower(Packet* p)`: must be provided by the transmitting PHY to determine the transmission power to be used for a given PLT.
- `getNoisePower(Packet* p)`: must be provided by the receiving PHY to determine the noise power at the receiver for a given PLT.
- `getTxAntenna(Packet* p)` and `getRxAntenna(Packet* p)`: must be provided by respectively the transmitting and receiving PHY to determine the antenna being used for a given PLT.
- `getTxSpectralMask(Packet* p)`: must be provided by the transmitting PHY to determine the spectrum used by a PLT.
- `getRxSpectralMask(Packet* p)`: must be provided by the receiving PHY to determine the RF filter used for receiving a PLT.
- `getModulationId(Packet* p)`: must be provided by the transmitting PHY to determine the modulation and coding scheme to be used for a PLT.

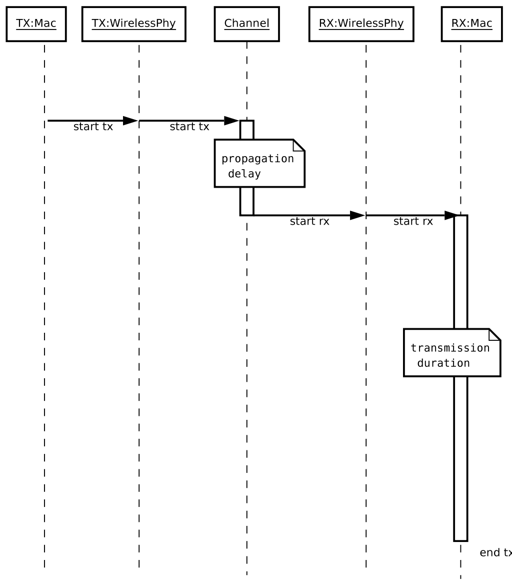

Figure 2: Sequence diagrams for packet TX/RX events in the MobileNode.

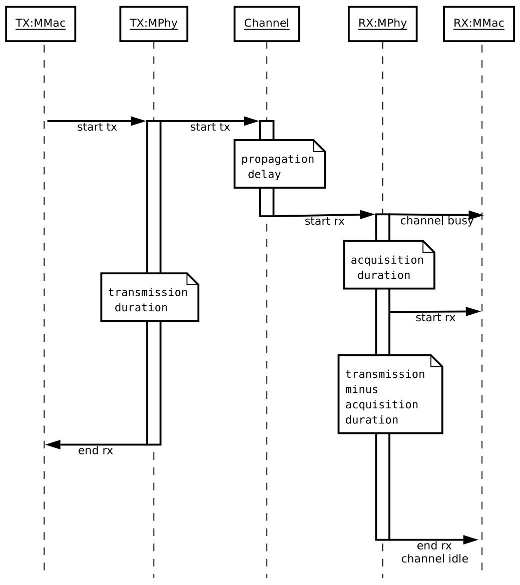

Figure 3: Sequence diagrams for packet TX/RX events in the MPhyMac.

- `startTx(Packet* p)`: the entry point for code to be executed at the beginning of a transmission. The implementation of this method is responsible for actually sending the `Packet` instance on the channel.
- `endTx(Packet* p)`: the entry point for code to be executed at the end of a transmission.
- `startRx(Packet* p)`: the entry point for code to be executed at the beginning of a reception. This code should handle the PLT acquisition process, e.g., implementing preamble detection, synchronization, etc.
- `endRx(Packet* p)`: the entry point for code to be executed at the end of a reception. This code is responsible for determining the presence of errors in the packet, using an error model suitable for the PHY technology being implemented, and for the eventual forwarding of the `Packet` instance to the upper layers.

One of the reasons for which we developed APIs for PHY and MAC layer development is that the ns2 `MobileNode` did not provide natively support to simulate the different phases of transmission and reception of packets, neither at the MAC nor at the PHY layer. This is represented in Figure 2: at the transmitter, only the beginning of a transmission is considered, both at the MAC and PHY layer; at the receiver, the PHY layer is only aware of the beginning of the reception, while the MAC layer has notion of both the beginning and end of the reception. This in our opinion is not a good design. First of all, whenever the MAC and the PHY layer need to perform any operation upon packet termination (i.e., change the status of the PHY or the MAC state machine), dedicated events need to be generated. Secondly, the duration of a transmission is determined at the PHY layer, since it depends on the packet size, the modulation and coding scheme, and possibly other PHY-specific aspects such as the length of synchronization preambles; consequently, having to determine it at the MAC layer to schedule the necessary events involves the duplication at the MAC layer of PHY layer attributes and functionalities, which can lead to inconsistencies and poor readability and maintainability of the code. Finally, this design has led to the misplacement of the implementation of several functionalities; for example, this is the case of PHY error models, which in several implementations had to be placed within the `recv_timer()` method of the MAC code.

Our design, represented in Figure 3, attempts to resolve these issues. First of all, the duration of a PLT is always determined by the PHY layer; furthermore, the scheduling of the start/end of transmissions and receptions events is a functionality provided by the MPhy base class, which takes care of calling the entry points for the technology-specific PHY layer code. Third, a set of cross-layer messages (functionality natively provided by the NS-Miracle framework) is defined so that MPhy-derived classes can trigger the transmission/reception start/end events on the MAC layer. Finally, the base MMac class defines some methods (`Phy2MacEndTx()`, `Phy2MacStartRx()` and `Phy2MacEndRx()`) which are called upon reception of the above mentioned messages, and can therefore be used by class inheriting from MMac to implement protocol-specific code which needs to be executed in response to the corresponding events. We note that our design is significantly closer than the `MobileNode` to the way in which real devices operate, for the same reasons which are discussed in [9] for the particular case of 802.11. This particular feature makes the Miracle PHY and MAC framework suitable for emulation.

2.3 Implemented modules

To conclude this section, it is to be mentioned that several types of wireless technologies have been implemented using the Miracle PHY and MAC framework. A first set of modules implements very generic technology such as a BPSK-based PHY layer and a ALOHA-based MAC; these modules were developed mainly as a proof of concept and for debug purposes, but still the fact that they have been implemented using the Miracle PHY and MAC framework provides them with features that, while rather trivial, could not have been easily implemented in other network simulators.[2] A

[2]For instance, the fact that the communication rate provided by the BPSK PHY is proportional to the spectrum that is assigned to it, and that the ALOHA MAC adapts its transmission rate to the com-

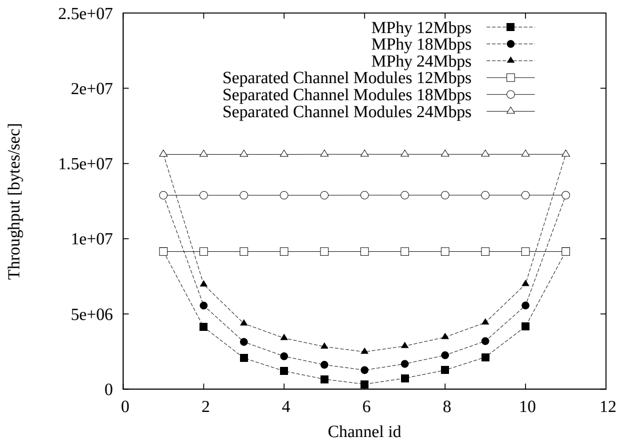

Figure 4: Throughput perceived at application layer.

second set of modules provides implementations of standard wireless technologies, including: 1) IEEE 802.11g, which we implemented by adding the necessary functionality to the dei80211mr library [8]; 2) UMTS release 4, which we implemented from scratch; 3) WiMAX, which was developed by a third party [12]. Finally, other modules have been implemented to support research activity not related to standard wireless technologies; the most relevant set of modules in this category is a collection of channel, PHY and MAC layer modules designed for Underwater communications.

2.4 Performance evaluation

As an example, in this section we describe one of the cases which highlights the features introduced by our framework. The IEEE 802.11g PHY standard [13] states that the channels which can be used for communication have a nominal bandwidth of 22 MHz, and are 5MHz apart from each other; as a result, inter-channel interference is expected when the channels being used are partially overlapping in frequency, and it has been reported to be a concern even for nominally orthogonal channels when the transmitter and receiver antennas are very close [14]. In literature, several models are known to evaluate inter-channel interference [15–17]; however, state-of-the-art simulators do not provide easy means of including those models in the simulation, since they lack the spectrum awareness that we discussed in the introduction. Thanks to the enhanced PHY and channel modeling support provided by our framework, and in particular to the usage of the SpectralMask class hierarchy, it is straightforward to implement any of the above mentioned inter-channel interference models. A comparison of all the existent inter-channel interference models would be beyond the scope of this paper; as a consequence, for the purpose of this example, we implemented only the adjacent channel attenuation model in [15], and used it to evaluate inter-channel interference using the well-known gaussian interference model. We considered a scenario in which two pair of nodes, each one consisting of a transmitter and a receiver, communicate simultaneously. The first pair uses always channel 6, while the other pair uses for every simulation a different channel chosen between 1 and 11. For both pairs, the transmitter sends packets at the maximum rate allowed by the MAC layer. Figure 4 shows how performance degrades as the channels used by the two pairs get closer.

munication rate of the underlying PHY without having to know how it is calculated, just by receiving a notification upon transmission end.

3. MROUTING

The coexistence of several radio interfaces in the same node poses a new research issue: how to actually make use of them. In recent literature there has been a significant number of publications on this topic; for instance, a well investigated problem is access selection in heterogeneous wireless networks. In [18] the authors study QoS provisioning in Always Best Connected (ABC) networks from architectural point of view. The work in [19] gives a comprehensive treatment of the mechanisms that are to be orchestrated for the realization of handovers at the IP level in heterogeneous and wireless networks. Finally, reference [20] evaluates the performance, in terms of load balancing, of several access schemes for the provisioning of voice over IP. We note that only in [20] we have an evaluation of the schemes proposed, and furthermore it is made by means of an analytical model, thus based on simplifying assumption in particular with respect to the wireless technologies being used. Due to high complexity of the architectures proposed, it would be very interesting to consider a deeper performance analysis of these solutions. Unfortunately, the lack of support for heterogeneous network scenarios in state-of-the-art network simulators has so far inhibited this type of evaluation. The NS-Miracle simulator [6] already offered the multi-interface and multi-technology capabilities needed for heterogeneous scenarios; what was missing was a generic routing scheme to jointly manage several radio interfaces. This has been the reason for the development of the MRouting framework.

Before starting the design process, we considered several issues typical of new research scenarios that had arisen during the evaluation of the Ambient Networks architecture [6, 21]. We would like to note that for the work within that project a specifically-designed routing solution had been used; the experience gained in this activity was fundamental in the subsequent design of MRouting. First of all, we noticed that we often have to deal with interfaces belonging to different technologies, and thus the relative routing schemes can differ a lot. In a typical scenario we often find in literature, we have mobile terminal equipped with both 802.11 and UMTS. In these cases, routing is usually achieved by combined an ad-hoc routing scheme for the 802.11 interface, and a static routing for the UMTS interface, both of them jointly managed by a single module in a two-tier structure. Such kind of scenario can easily become more complex, e.g., by integrating other technologies such as WiMAX, or by incrementing the number of interfaces per technology. In these scenarios, the decision of which interface to use becomes more complex, especially considering that the technology-specific metrics, which are commonly used for routing purposes, are often not suitable for use in a multi-technology scenario, due to difficulties in comparing metrics of different types.

The MRouting framework is designed to formalize the way to solve both these problems in order to make it easier to develop hybrid routing schemes in complex heterogeneous network scenarios. In the following subsection, we analyze a typical use case in order to pinpoint both the issues to be solved and how our solution works. Then, we will describe how we ported a standard ns2 routing algorithm implementation, the Ad hoc On Demand Distance Vector (AODV), to the MRouting framework; in doing this, we will take the opportunity to describe the MRouting API.

3.1 Framework specification

The design principles of our MRouting framework are more easily explained by an example. We consider a scenario in which Mobile Terminals (MTs) are equipped with three radio interfaces, two IEEE802.11 and one UMTS, and want to communicate with a fixed host in the internet through one of these wireless interfaces. The

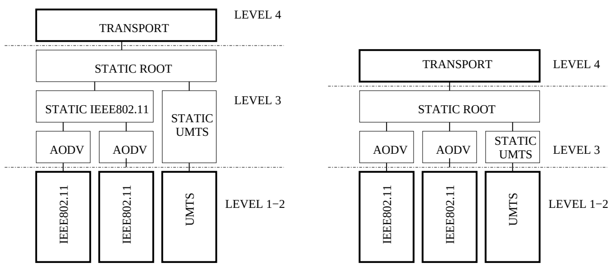

Figure 5: Diagram of the example routing architectures: three-tier architecture (left) and two-tier architecture (right)

two 802.11 interfaces provide access to the internet through two ad-hoc networks, on each of which the AODV protocol is used, while the UMTS network provides direct access to the internet and needs static routing only. MTs may exploit one interface at a time, with decisions performed on a per-packet basis. We note that, from the point of view of the technology, this is a scenario that is already feasible with the commercial equipment available nowadays, but its full investigation by means of state-of-the-art simulators would be rather challenging.

Considering MTs, the first problem we faced was how to define a structure which encapsulates all the routing schemes of the available interfaces, keeping in mind that they might be using different wireless technologies, and therefore also their routing modules might be different. Moreover, we might want to exploit different routing algorithms even for interfaces using the same radio technology (e.g., in order to increase the diversity of the paths available) or to use the same routing module for several interfaces using different radio technologies. The latter is for instance the case of some schemes which have been proposed in literature, such as Cisco Administrative Distance and Zebra [22]. We would like to stress that MRouting is a framework which tries to formalize the definition of a multi-tier routing architecture and not a routing scheme itself, therefore it is left to the final developer to define the particular architecture being used. This choice was made to leave as much freedom as possible to the final developers, so that a wide range of algorithms can be simulated.

The basic block of MRouting is the `MrclRouting` class, which is a child of the Miracle `Module` class. This class is in charge of actually containing all the procedures implemented in the routing algorithm. Every particular routing module is implemented by extending the `MrclRouting` class. All routing modules within the same node are arranged in a tree structure. In this way, the leaves of the tree can be made up by a dedicated routing scheme for each interface, while other nodes in the tree can be exploited to jointly manage all underlying modules in a hierarchical fashion. These other nodes, in particular, can also provide forwarding functionality, not only between different interfaces, but also between different networks implementing different routing protocols. In the left part of Figure 5 we describe a first possible solution for the example scenario involving 802.11 and UMTS that we introduced before. A three-tier tree architecture in the routing layer is used: the first level (i.e., the leaves of the tree) is dedicated to manage the particular radio interface, the second level is in charge of jointly managing the interfaces belonging to the same radio technology and, finally, the root of the tree acts as a decision engine selecting among different radio technologies. We might also define a simpler scenario in which all the radios of the same technology are managed by the same routing module, which has to directly manage all the available radio interfaces. In this case, represented on the right side of Figure 5, we have a two-tier tree structure

In the following, we refer to *routing layer* as the set of all the routing modules in the architecture (i.e., in all the tiers). We would like to observe that, using the cross-layer message functionality provided by the NS-Miracle framework, the routing layer becomes automatically aware of its own architecture at run-time.

In fact, it is the routing module framework which by itself discovers the structure implemented when it has to determine how to forward a packet. In this situation, independently from the direction of the flow (i.e., packets from any interface or from the upper layer), the first routing module which has to process the packet, propagates a cross-layer message to all the other modules within the routing layer in order to get comprehensive information on how to route the packet and who can forward it. During this flooding process, a twofold learning is performed. First of all, the actual structure of the routing layer is discovered, and the number of possible solutions (i.e., the number of leaves of the tree) is determined. Secondly, all the informations to be used to make the decision on which interface will be exploited are collected. With this respect, a fundamental issue is what information is to be used to perform the routing decision. Practically all routing schemes rely on a cost function to select the best route. The problem is that different metrics are commonly used for different routing schemes (e.g., hop count, round trip time, delay, jitter, energy, cost, business relationship, etc.), and in general heterogeneous metrics cannot be compared among themselves. To solve this issue, we introduced the `Metric` class, which is an interface-class to be extended by all the metrics to be used in the framework. Thanks to this class, it is possible to define the particular routing metrics we want to consider in the scenario and to use them dynamically in the structure. When looking for the possible routes, a list of the available paths is returned, sorted by the metric values in descending order. To handle the case in which several routing algorithms using different metrics need to coexist and to be compared among themselves, we introduced the definition of composite metrics, i.e., metrics which can be assigned a value by convertion from other metrics, where the conversion policy can be specified to fit the particular scenario being considered. For instance, in our sample scenario, the AODV routing protocol being used for the 802.11 exports the hop count as routing metric, whereas UMTS has an associated monetary cost per usage. In this particular case, we performed metric conversion by assigning a monetary cost for every hop in the 802.11 network. The resulting routing policy was to route packets over UMTS if its cost is lower than any available 802.11 route, otherwise use the 802.11 route having the lowest cost. We note that, thanks to the `Metric` class hierarchy, implementing more complex policies possibly based on different metrics would have been straightforward. However, we have also to observe that we left to the final developer the choice of how to combine metrics. In fact, in order to not limit our framework, we only provide the policy that all the metrics collected are summed and then sorted in ascending order, therefore is a matter of the developer to find the proper codification of them in order to implement the desired access selection policy. This is due to fact that each radio technology has its specific metrics and it's a matter of the decision making engine policy to give priority to each of them, this makes very hard the definition of a general engine to manage this problem and this is the only general solution we found.

3.2 MRouting AODV

In this section we give an overview of the steps we followed to port the ns2 AODV routing scheme implementation, and the novelties we introduced thanks to MRouting. In doing this, we will

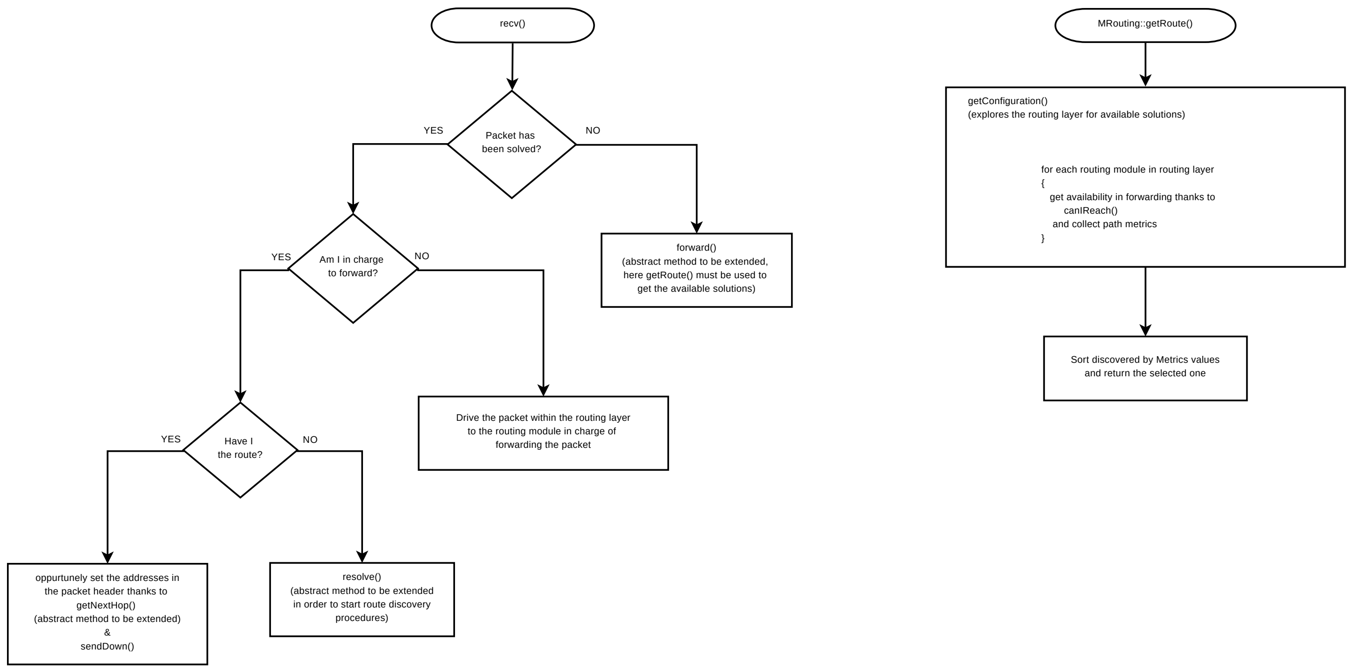
recv()
Packet has been solved?
YES
NO
Am I in charge to forward?
YES
NO
forward() (abstract method to be extended, here getRoute() must be used to get the available solutions)
Have I the route?
YES
NO
Drive the packet within the routing layer to the routing module in charge of forwarding the packet
oppurtunely set the addresses in the packet header thanks to getNextHop() (abstract method to be extended) & sendDown()
resolve() (abstract method to be extended in order to start route discovery procedures)
MRouting::getRoute()
getConfiguration() (explores the routing layer for available solutions)
for each routing module in routing layer { get availability in forwarding thanks to canIReach() and collect path metrics }
Sort discovered by Metrics values and return the selected one

Figure 6: Flowchart diagram of the MRouting recv (left) and getRoute (right) methods.

also take the opportunity to describe the MRouting API, in order to give an overview of how it can be used to implement new routing modules or to port existing ones.

First of all, we want to note that it is possible to simply wrap the standard ns2 AODV module as-is in a Miracle Module class (adopting the procedure described in [6]), but we decided to fork the AODV code in order to allow the full integration with our framework and provide enhanced functionalities. The classic receive method (i.e., `recv()`) has been modified and formalized for the whole framework, and it is internally managed according to the flowcart diagram in the left part of Fig. 6. This implies that the standard code must to be adapted to a set of API defined in MRouting, i.e.:

- `controlPacket(Packet* p)`: this method is in charge of recognizing internal routing packets and processing them. In our specific case, it has only to call the old `recvAodv()` method, in which the algorithm processes its control packet (i.e., Route Request, Route Reply, Route Error and Hello messages).

- `canIReach(Packet* p)`: this method has to return the availability of forwarding to the requested destination and the relative metric values; it is used internally by the MRouting framework to discover all the solutions available for a certain destination. In this particular case, it is implemented as a query to the internal AODV routes table, in which we provide the support to manage several metrics.

- `forward(Packet* p)`: this is the method called each time a new packet arrives in the routing layer. Therefore, it is in charge of discovering all the available solutions within all the routing modules in the routing layer and making a decision on which route to use (i.e., by selecting the route having the lowest cost according to the desired metric). To do this, it has to exploit the `getRoute(Packet* p)` method, provided by the framework, which propagates the request to the whole architecture, and returns an ordered list containing all the modules that can forward the packet, as depicted in the right part of Fig. 6; the list is ordered with respect to the metric that is being considered.

- `resolve(Packet* p)`: this method is in charge of solving the packet in the particular case that the module is requested to forward the packet, but the path is unavailable. This method is used in ad-hoc routing to manage routes dynamically and handle possible failures. In AODV, this method discovers new paths and queues the packet to be forwarded.

- `getNextHop(Packet* p)`: this method is internally called when it has to forward a packet via this particular extension of the routing module, therefore it has to know how to reach the destination.

Thanks to the MRouting framework, when AODV receives a packet it has a comprehensive view of the routing architecture, and can obtain all the available solutions to forward the packet, even those belonging to other interfaces. Therefore AODV may select which route is better to use as function of the metric defined (or the combination of them). This makes it possible to change the interface when it is necessary (e.g., to extend service coverage among different subnetworks potentially using different technology). This is done also during the route discovery process: in this case an AODV module which receives an Route Request packet does not limit its view to its own route table but it asks to the whole routing layer to find the answer. Finally, we note that in spite of the additional functionalities introduced, our MRouting AODV implementation defaults back to the standard AODV behavior when a single interface is being used.

4. CONCLUSIONS

In this paper we presented a framework, developed as an extension of NS-Miracle [6], which has the purpose of addressing some significant issues found in state-of-the art network simulators concerning the modeling and simulation of the lowest three layers of the protocol stack. The Miracle PHY and MAC module provides support for enhanced channel modeling while at the same time addressing the issue of PHY and MAC code modularity and reusability. The Miracle Routing module formalized the definition of multi-tier routing to provide support for the simulation of routing schemes designed for multi-interface devices and heterogeneous networks. To conclude, the framework described in this paper, which is available at [23], is an effective tool for facilitating the simulation and the design of 4G networks, and candidates as a reference approach for the proper addressing of some of the key challenges in wireless network simulation.

5. ACKNOWLEDGMENTS

This article is dedicated to the memory of Federico Maguolo, who died in a tragic accident on June 24, 2008.

6. REFERENCES

[1] S. Shakkottai, T. Rappaport, and P. Karlsson, “Cross-layer design for wireless networks,” *IEEE Communications Magazine*, vol. 41, no. 10, pp. 74–80, Oct. 2003.

[2] I. F. Akyildiz and X. Wang, “Cross-layer design in wireless mesh networks,” in *IEEE Transactions on Vehicular Technology*, vol. 57, no. 2, March 2008, pp. 1061–1076.

[3] R. Thomas, D. Friend, L. Dasilva, and A. Mackenzie, “Cognitive networks: adaptation and learning to achieve end-to-end performance objectives,” *IEEE Communications Magazine*, vol. 44, no. 12, pp. 51–57, 2006.

[4] S. Haykin, “Cognitive radio: brain-empowered wireless communications,” *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 201–220, 2005.

[5] D. Clark, C. Partridge, J. Ramming, and J. Wroclawski, “A knowledge plane for the internet,” *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 3–10, 2003.

[6] N. Baldo, F. Maguolo, M. Miozzo, M. Rossi and M. Zorzi, “ns2-MIRACLE: a Modular Framework for Multi-Technology and Cross-Layer Support in Network Simulator 2,” in *ACM NSTools*, Nantes, France, Oct 2007.

[7] New ns-2 802.11 support, http://yans.inria.fr/ns-2-80211/.

[8] An improved 802.11 implementation for ns2 with enhanced interference model, http://www.dei.unipd.it/ricerca/signet/tools/dei80211mr.

[9] Q. Chen, F. Schmidt-Eisenlohr, D. Jiang, M. Torrent-Moreno, L. Delgrossi, and H. Hartenstein, “Overhaul of IEEE 802.11 modeling and simulation in ns-2,” *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pp. 159–168, 2007.

[10] L. Betancur, R. Hincapié, and R. Bustamante, “WiMAX channel: PHY model in network simulator 2,” *ACM International Conference Proceeding Series; Vol. 202*, 2006.

[11] A. F. Harris and M. Zorzi, “Modeling the Underwater Acoustic Channel in ns2,” in *ACM NSTools*, Nantes, France, Oct 2007.

[12] NS2MiracleWimax, http://ns2miraclewimax.sourceforge.net/.

[13] “Further higher-speed physical layer extension in the 2.4 ghz band,” IEEE Std. 802.11g, 2003.

[14] P. Fuxjager, D. Valerio, and F. Ricciato, “The myth of non-overlapping channels: interference measurements in IEEE 802.11,” *Proceedings of the Conference on Wireless on Demand Network Systems and Services (WONS)*, pp. 1–8, 2007.

[15] E. Villegas, E. López-Aguilera, R. Vidal, and J. Paradells, “Effect of adjacent-channel interference in IEEE 802.11 WLANs,” 2007.

[16] C. Cheng, P. Hsiao, H. Kung, and D. Vlah, “Adjacent Channel Interference in Dual-radio 802.11 a Nodes and Its Impact on Multi-hop Networking,” *IEEE Globecom*, 2006.

[17] E. Garcia, E. Lopez-Aguilera, R. Vidal, and J. Paradells, “IEEE Wireless LAN Capacity in Multicell Environments with Rate Adaptation,” *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, pp. 1–6, 2007.

[18] G. Fodor, A. Eriksson, and A. Tuoriniemi, “Providing Quality of Service in Always Best Connected Networks,” *IEEE Commun. Mag.*, vol. 41, no. 7, pp. 154–163, July 2003.

[19] W. Zhang, J. Jaehnert, and K. Dolzer, “Design and Evaluation of a Handover Decision Strategy for 4th Generation Mobile Networks,” in *Proceedings of IEEE VTC*, Jeju, Korea, Apr. 2003.

[20] A. P. da Silva, F. R. P. Cavalcanti and R. A. de O. Neto, “VoIP Capacity Analysis of Wireless Multi-access Networks using Access Selection Schemes,” in *Proceedings of IEEE PIMRC*, Athens, Greece, Sept. 2007.

[21] M. Miozzo, M. Rossi and M. Zorzi, “Architectures for Seamless Handover Support in Heterogeneous Wireless Networks,” in *IEEE WCNC*, Las Vegas, NV, US, April 2008.

[22] GNU Zebra, http://www.zebra.org/.

[23] ns-MIRACLE: Multi InteRfAce Cross Layer Extension for ns-2, http://telecom.dei.unipd.it/download.