WiMsh: A Simple and Efficient Tool for Simulating IEEE 802.16 Wireless Mesh Networks in ns-2

Claudio Cicconetti
Dipartimento di Ingegneria dell'Informazione
Via Diotisalvi, 2
56122 Pisa, Italy
c.cicconetti@iet.unipi.it

Ian F. Akyildiz
School of Electrical and Computing Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
ian@ece.gatech.edu

Luciano Lenzini
Dipartimento di Ingegneria dell'Informazione
Via Diotisalvi, 2
56122 Pisa, Italy
l.lenzini@iet.unipi.it

ABSTRACT

Wireless mesh networks (WMNs) are two-tier wireless multi-hop networks. The top tier is made of wireless routers, which provide access to the wireless clients in the bottom tier. One technology for enabling multi-hop communication in the top tier is IEEE 802.16, which includes a *mesh* mode, in addition to the Point-to-Multipoint mode for cellular networks. As is often the case with wireless networks, simulation is often employed as the primary means of investigation. There are several network simulation tools, both commercial and free-of-charge, with IEEE 802.16 PMP support. However, the MAC protocol designed for mesh mode is substantially different from that for PMP operation, which creates the need for a specific simulation tool. In this paper we describe a simulation module, called WiMsh, than enables simulation of IEEE 802.16 wireless mesh networks with the popular Network Simulator 2. We have made publicly available WiMsh in October 2007 as open source software.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation support systems—*environments*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*statistical software*; D.2.6 [**Software Engineering**]: Programming environments—*performance measures*

General Terms

Experimentation, Measurement, Performance

Keywords

wireless mesh networks, simulation, network simulator 2, IEEE 802.16

1. INTRODUCTION

During the last few years, wireless mesh networks (WMNs) have emerged as a flexible and cost-effective alternative to both wired and wireless infrastructure networks[6]. In a WMN the wireless mesh routers co-operate to create a backbone, which is used by the wireless mesh clients to access either the Internet or intra-WMN network services. There are several directions for research at the backbone tier, which have been so far investigated, including[2]: layer 2 (L2) routing and forwarding issues, scheduling and quality of service (QoS) provisioning, congestion control and load balancing, medium access control (MAC) protocols. Most research studies, as well as practical solutions, employ IEEE 802.11-equipped devices as wireless mesh routers. However, this standard does not specify how to enable the basic WMN functions, such as multi-hop forwarding and L2 routing. To fill this gap, the Task Group 's' was established to produce an amendment to the standard for enabling native support of the WMN functions in IEEE 802.11[18]. The first release of the IEEE 802.11s is expected before the end of 2009.

Another standard solution, which is available since 2004, is IEEE 802.16 standard[1], which includes *mesh* mode support. The MAC protocol for WMNs is different from and non-inter-operable with the point-to-multipoint (PMP) MAC protocol, which has recently gained popularity due to its applications to fixed and mobile Broadband Wireless Access (BWA) cellular networks. Therefore, all the findings obtained for IEEE 802.16 PMP do not apply to IEEE 802.16 mesh. In the former, i.e. the PMP mode, all the decisions about medium access are taken centrally by the Base Station (BS). On the other hand, in mesh mode, there are two scheduling schemes: centralized and distributed. With centralized scheduling, nodes are arranged in a logical tree topology. The BS is the root of the tree and it schedules bandwidth on all the links in a centralized manner[8]. This approach makes it easier to control the network dynamics, because all decisions are taken by a single node. However, the signaling overhead, which is needed to provide the BS with complete knowledge of the network, is rather high, in terms of both latency and consumed bandwidth. On the other hand, with distributed scheduling, all nodes take their decisions locally. The standard specifies a set of mechanisms to ensure that data are transmitted in a collision-free manner within any neighborhood of nodes. Spatial re-use can be exploited by nodes that are at least two hops away. Medium access for control messages is modeled and its performance is analyzed in [7], while a scheduling algorithm

SIMUTools 2009 Rome, Italy.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

to assign bandwidth in a fair manner is proposed in [10]. The distributed mode is more flexible and efficient than the centralized mode, since it follows more closely network variations, like channel quality and traffic conditions.

The performance studies of WMNs are typically carried out by means of packet-level simulation. In fact, accurate analytical model of wireless networks are often too complex, and realizing a prototypical test-bed is a cost- and time-consuming solution, which might not be feasible in the early stages of development. Unfortunately, IEEE 802.16 mesh is not included in most popular network simulators, like Opnet[1], Qualnet[2], ns-2[3], Omnet++[4], only support IEEE 802.11 and IEEE 802.16 PMP. For this reason, in 2006 we developed a simulation module for IEEE 802.16 mesh with distributed scheduling, called WiMsh. We have selected ns-2 as the simulation environment because it is widely accepted by the networking research community, and it is currently being re-designed to improve efficiency and simulation accuracy as ns-3[14]. We have used WiMsh to carry out several simulation campaigns, whose results have been presented in high-level IEEE- and ACM-sponsored conferences, including [9, 11]. WiMsh has been released under the GNU Public Licence (GPL) in October 2007[5], as a patch to ns-2. No modifications to any module operating at layer 3 or above are required to run simulations. Since its release, the module has been downloaded 2000+ times by students and researchers from several countries, and it is currently used by many of them for their research.

At the time of writing, to the best of our knowledge, the only network simulator that includes support for IEEE 802.16 mesh is NCTUns[17]. This tool is well-designed and has been shown to produce accurate results in many scenarios. However, our module has some advantages with respect to NCTUns, as far as the IEEE 802.16 mesh is concerned. First, the architecture of the IEEE 802.16 mesh sub-system has not been described by the authors of NCTUns, which makes it difficult to operate with it, especially if the user wants to perform complex operations, like implementing a new scheduling algorithm. Second, the penetration of NCTUns in the scientific community is not comparable to that of ns-2. One of the reasons why ns-2 is so widely used is that it runs (virtually) on any operating system, while NCTUns requires a specific Linux distribution (Fedora 9) to run smoothly. Finally, ns-2 includes a vast amount of applications and transport layer agents, which can be re-used seamlessly by WiMsh. NCTUns, instead, only provides a limited set of traffic models. In part, this is expected because NCTUns has a strong inclination for network *emulation*, rather than *simulation*, which however requires special skills and can only be carried out in limited scenarios.

The rest of this paper is organized as follows. In Section 2 we describe the aspects of the MAC and physical layer of IEEE 802.16 mesh with distributed scheduling that are implemented in WiMsh, which is described in Section 3. In Section 4 we explain how to configure simulation scenarios in an easy and flexible manner, and also discuss the use of WiMsh for educational purposes. We conclude the paper in Section 5.

[1]http://www.opnet.com/
[2]http://www.scalable-networks.com/
[3]http://www.isi.edu/nsnam/ns/
[4]http://www.omnetpp.org/
[5]http://cng1.iet.unipi.it/wiki/index.php/Ns2mesh80216

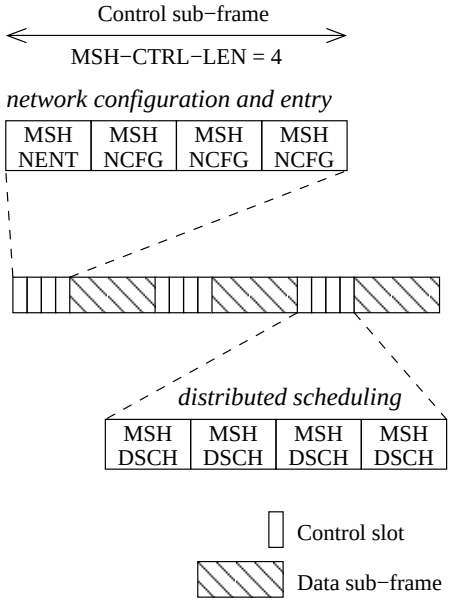

Figure 1: Example of frame structure.

2. IEEE 802.16 MESH WITH DISTRIBUTED SCHEDULING

An IEEE 802.16 WMN network[1] with distributed coordinated scheduling consists of a Base Station (BS) and many Subscriber Stations (SSs). Unlike the Point-to-Multipoint (PMP) mode, the Mesh mode allows SSs to communicate directly, without relaying their transmissions through the BS. Medium access is coordinated in a distributed fashion. The BS, in particular, has to coordinate with its neighbors as any other station. Its only special functions are: advertising the network configuration and authenticating a new SS when entering the network. Hereafter, we do not distinguish between the BS and SSs, and refer to both of them as *nodes*. Furthermore, we define two nodes as *neighbors* if they can communicate directly with each other. Two nodes that are not neighbors, but share a common neighbor, are said *two-hop neighbors*.

The MAC protocol is frame-based: all nodes are synchronized and they transmit data and control messages in slots of fixed duration over non-interfering channels, up to 16. Orthogonal Frequency Division Multiplexing (OFDM) is used for transmission. Each frame consists of a control sub-frame and a data sub-frame, as illustrated in Fig. 1. The control sub-frame is made of MSH-CTRL-LEN control slots, each consisting of seven OFDM symbols. The number of data slots, which create the data sub-frame, and their duration depend on the physical network configuration parameters. The values specified in the standard physical profiles are reported in Table 1. The number of bytes that can be conveyed in a slot depends on the Modulation and Coding Scheme (MCS) used. The MCSs specified by the standard are reported in Table 2. When there is more than one channel available, data can be transmitted by different nodes at the same time on different channels with no reciprocal in-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

Bandwidth MHz	Frame duration ms	Slots per frame
3	10	98
3.5	4	30
5.5	10	204
7	4	90
10	4	141

Table 1: Number of data slots per frame, with 4 control slots and cyclic prefix equal to 1/16.

Index	Modulation	Coding rate	Bytes/slot
0	QPSK	1/2	24
1	QPSK	3/4	36
2	16-QAM	1/2	48
3	16-QAM	3/4	72
4	64-QAM	2/3	96
5	64-QAM	3/4	108

Table 2: Number of bytes conveyed per slot with different MCSs. Index is used in WiMsh to identify the MCS.

terference. However, messages in the control sub-frame are always transmitted in the same channel, which is known to all nodes in the network.

The slots in the control sub-frames are used by nodes to broadcast the mesh distributed scheduling (MSH-DSCH) messages, which are used for negotiating access to the data sub-frame. All nodes run a distributed election procedure specified by the standard to transmit regularly these messages in a collision-free manner. For a node, the average interval between two consecutive opportunities to send a MSH-DSCH message depends on the number of its neighbors and the value of the system parameter *XmtHoldoffExponent* [7].

One control sub-frame is allocated periodically for network configuration and maintenance only. Two messages can be transmitted in these sub-frames: mesh network entry (MSH-NENT), which are used by new nodes willing to associate with the WMN; and, mesh network configuration (MSH-NCFG), which disseminate the network parameters to be employed and provide nodes with a means to setup uni-directional links between them. As shown in Fig. 1, in the control sub-frame reserved for this use, the first control slot is employed for network entry, while the remaining MSH-CTRL-LEN - 1 slots are occupied by MSH-NCFG messages. The latter, i.e. MSH-NCFG messages, are transmitted in a collision-free manner, like MSH-DSCH messages but using a separate instance of the distributed election procedure. In a steady state, by exploiting the network configuration data received in the past, any node acquires the complete topology of its two-hop neighborhood. On the other hand, MSH-NENT messages are transmitted with no coordination, hence two or more MSH-NENT messages can collide together. Should this happen, a collision resolution procedure specified by the standard is used to restore the network entry process[6]

The following countermeasures are taken to limit the chance that the control messages are lost because of channel errors, e.g. due to short-term fading variations: the messages are transmitted using the most robust MCS, i.e. QPSK with coding rate 1/2; a long preamble, i.e. two OFDM symbols, is prepended to each message; and, the last OFDM symbol is left unused as a guard time.[7]

Data transmission is coordinated in a distributed manner by means of the following information elements (IEs) contained in MSH-DSCH messages. Since MSH-DSCH messages are broadcast, and a single channel is used for the control sub-frame, all the nodes in a neighborhood receive them. Thus, the IEs in the same MSH-DSCH can be directed to different recipients. There are four IE types: request, grant, confirmation, and availability. The *request* IE is used by a node, called requester, to ask a neighbor, called granter, to allocate some bandwidth, in terms of slots, to it. When the granter transmits a MSH-DSCH, it can send a *grant* IE to the requester, which contains a range of slots over a range of frames in a given channel. This three-way handshake is complete when the requester sends a *confirmation* IE to acknowledge that it will actually send data in the granted slots. Finally, *availability* IEs can be used optionally by nodes to indicate certain sets of slots that cannot be used to transmit or receive data, e.g. because of a partially overlapping grant of a neighbor in the same channel. The algorithm for scheduling bandwidth, i.e. filling the MSH-DSCH messages with IEs, is left unspecified by the standard.

When a node confirms a grant, it will eventually transmit MAC Protocol Data Units (PDUs) in the range of the slots granted, using the channel specified by the granter. A single MAC Service Data Unit (SDU), e.g. an IPv4 datagram, can be fragmented into many MAC PDUs, and multiple MAC SDUs can be packed together to form a single MAC PDU. Both mechanisms can be used to reduce MAC overhead or improve transmission efficiency. Furthermore, the IEEE 802.16 mesh MAC introduces the opportunity for a node to set the following two fields of the MAC PDU header: 2-bit *drop precedence* field, which can be used to enable different dropping policy at congested nodes; 3-bit priority field, which can be used by nodes for service differentiation.

3. SIMULATOR DESCRIPTION

In this section we describe the WiMsh simulator[8]. A blueprint of the whole IEEE 802.16 mesh sub-system is illustrated in Fig. 2. Every module represented in the figure is fully implemented via a C++ class. Interactions between modules are classified into two types: solid lines represent packet exchanges, while a dashed line between two elements means that they communicate via their respective Application Programming Interfaces (APIs), i.e. C++ member functions. The relevant parameters of each module can be easily configured by the user, as described in detail in Section 4.

The main module of WiMsh is the MAC layer, which is connected to the Link Layer (LL) ns-2 module. The LL has been modified only slightly, since it already acts as a

[6]The network entry latency can be evaluated with WiMsh. Due to limited page budget, we do not describe in detail this simulation scenario, which is of minor interest with respect to the case when the network is in a steady state.

[7]MSH-NENT messages use the last *three* OFDM symbols as guard time.

[8]As a naming convention in the source code, all the names of data types have been prepended with "WiMsh", to avoid ambiguity with existing ns-2 object types. However, this particle is omitted in the paper for the sake of readability.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

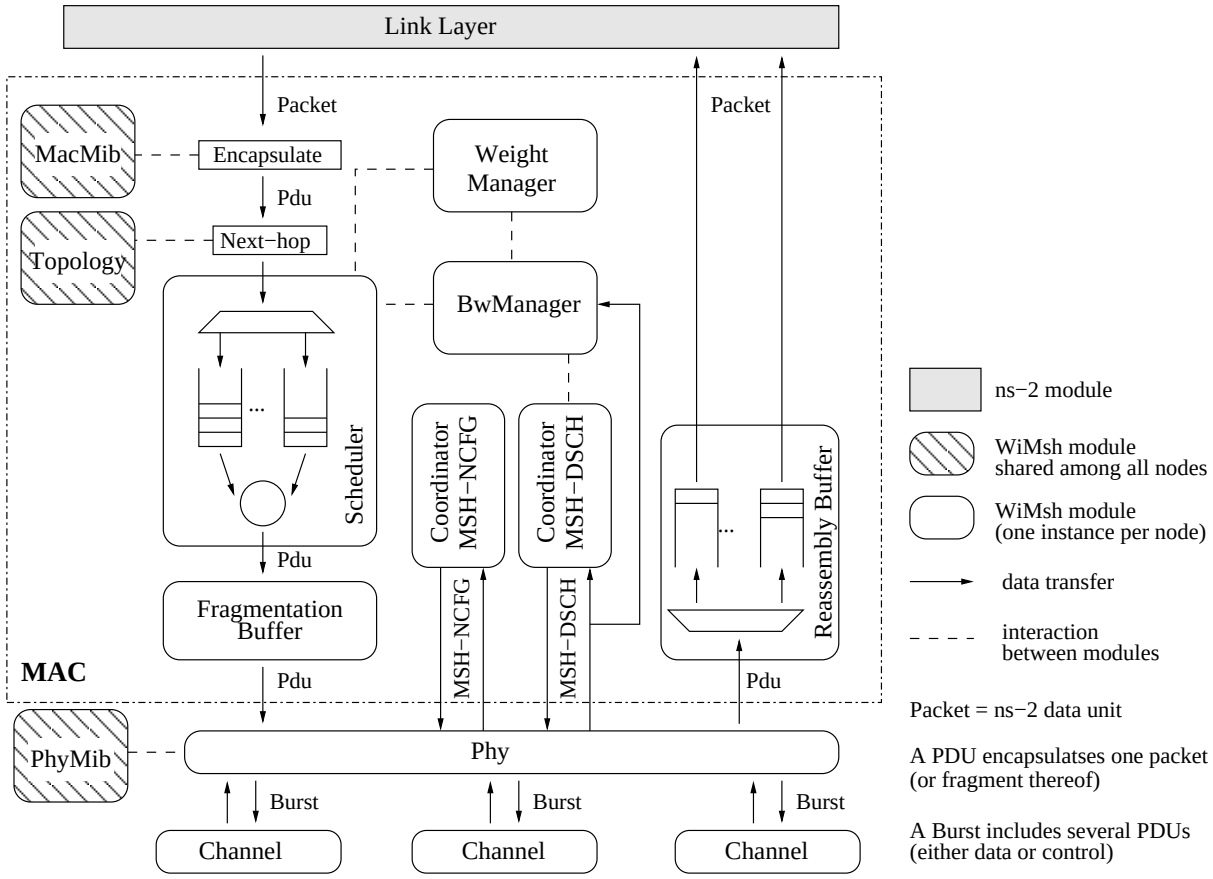

Figure 2: The WiMsh architecture.

two-way connector between the node's agents and the MAC object. No changes have been applied to other ns-2 modules, such as transport layer agents and applications, which are, thus, fully inter-operable with WiMsh. The MAC module is divided into several sub-modules. Every module is defined as a separate C++ class, which makes it easier to extend its functionalists by exploiting C++ inheritance and polymorphism. While the MAC protocol is modeled in detail, so as to evaluate accurately the impact of its mechanisms and algorithms on the performance, a simplified model of the physical layer is currently included in WiMsh. In fact, in the classical application scenarios of WMNs, the nodes are fixed. Sometimes, their position is carefully planned by the network operator, using procedures similar to those for locating base stations and repeaters in cellular networks. Therefore, we expect the response of the wireless channel to be much more stable and predictable than those of, e.g., mobile or vehicular ad-hoc networks. However, the physical layer and channel modules have been designed so as to allow more sophisticated propagation models to be integrated.

Due to limited page budget, the implementation details will be left out of our discussion. We refer the interested reader to the comprehensive documentation automatically generated via Doxygen[9] from the inline comments in the C++ header files, which can be found in the WiMsh website in both HTML and PDF.

[9]http://www.doxygen.org/

3.1 Packets

In ns-2 there exists one single transfer unit type, for all layers, called *Packet*. When a packet is passed from one layer to another, encapsulation is modeled by filling the fields in the relevant header. In other words, any packet includes room for all the headers of all the protocols supported by ns-2. Therefore, no memory copy/extraction for encapsulation/de-encapsulation actually happen in practice, which leads to efficient simulations, in terms of run time. However, this design choice is often cause of confusion, and also leads to error-prone coding. For instance, if the developer "forgets" to fill some fields of the relevant header, their values are uninitialized, but assumed to be set correctly by the underlying layers. Such an error is impossible to detect using static debugging tools, because the compiler cannot distinguish a field wrongly uninitialized from another that is simply not used in the protocols encapsulated so far. Therefore, we decided to follow a different path, described in the following.

As soon as an ns-2 packet is passed from the LL to the WiMsh MAC, it is encapsulated into a dedicated structured, called *Sdu*, which models the MAC SDU, and also contains some internal variables for collecting packet-level statistics.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

Further classification is performed when the *Sdu* is encapsulated into a *Pdu* object, which models the MAC Protocol Data Unit (PDU) and contains the MAC header (*MacHeader*). Among the other fields, the MAC header contains the IEEE 802.16 mesh sub-header, which includes the priority and drop precedence of the packet. These fields are set according to the MAC Management Information Base (MIB) object (*MacMib*), which is shared among all the nodes in the network. In reality, the MAC MIB is configured by the network operator and installed on nodes using the Simple Network Management Protocol (SNMP). The MacMib can be configured by the user so that all the packets with a given ns-2 traffic flow identifier are assigned a given priority and drop precedence.

In addition to the data packets, there are three MAC control messages implemented: MSH-DSCH, MSH-NCFG, and MSH-NENT, described in Section 2. Both data and control messages are encapsulated into a physical layer transmission unit, called *Burst*, which is passed to the physical layer for over-the-air delivery to the peer MAC object(s). A Burst can contain only a single control message, while multiple MAC PDUs can be added, in accordance with the standard.

3.2 MAC

In this section we describe the MAC sub-modules.

3.2.1 Coordinator

The *Coordinator* is the module that executes the distributed election procedure. For this reason, two instances of this module exist: one for distributed scheduling and another for network configuration. These instances are passed from the physical layer the respective messages handled. When a MSH-DSCH or MSH-NCFG message is created, encapsulated into a Burst, and passed to the physical layer, the control slot for sending the subsequent one is decided. With MSH-DSCH only, the Coordinator queries the bandwidth manager *BwManager* object to obtain the request, grant, confirmation, and availability IEs to insert into the outgoing MSH-DSCH.

3.2.2 BwManager

The BwManager is the core component of the MAC protocol, since it runs the algorithm for bandwidth scheduling. The bandwidth manager included in WiMsh is FEBA, which has been shown to provide fairness and service differentiation in varied network and traffic configurations[10]. FEBA has several configuration parameters that can be used to tune its behavior, e.g., to relax (or disable) fairness provisioning. In FEBA a traffic flow is uniquely identified by the triple $\langle s, d, p \rangle$, where s is the source node, d is the destination node, and p is the priority. Unlike its PMP counterpart, the IEEE 802.16 mesh MAC is connection-less, i.e. it does not specify a procedure to establish a traffic flow between two nodes. Therefore, we created a dedicated module, called *WeightManager*, to keep track of all the traffic flows traversing a node. Traffic flows are implicitly torn down when no more packets with matching triple are received for a timeout period. This module also keeps updated the numeric *weight* of traffic flows, based on a configurable priority value. For instance, if three flows are traversing a node, and their priority values are 1, 1, and 2, then the first two flows will have weight $1/(1+1+2) = 1/4$ and the last one $2/(1+1+2) = 1/2$.

To enforce collision-free scheduling, the BwManager must avoid granting or confirming slots that have been already reserved or indicated as unavailable by other nodes. To this aim, it keeps an up-to-date timetable of the grants assigned in the subsequent slots, based on the MSH-DSCH that it is passed by the Phy. In WiMsh we defined a data structure called *neigh_tx_unavl*, hereafter NTU for brevity: $NTU \in \{0,1\}^{N \times C \times F \times S}$, where N is the number of the node's neighbors, C is the number of channels, F is the maximum scheduling horizon, and S is the number of data slots per frame. The element $NTU(n, c, f, s)$ thus contains 1 only if the n-th neighbor cannot be granted or confirmed slots, in channel c, in the s-th slot of frame f (modulo F). NTU is implemented as a multi-dimensional vector of bitsets to enable fast random access. This data structure is accessed: at the beginning of every frame to eliminate the data of the past frame, which are not needed anymore; every time a MSH-DSCH message is received from a neighbor; during bandwidth scheduling to determine which slots can be granted, confirmed, or indicated as unavailable. Many utility functions are provided in the source code to perform such operations efficiently.

3.2.3 Scheduler

When a grant has been confirmed by the requester node, the slots in the range can be actually used for data transmission, expect the first OFDM symbol that is needed to transmit a short preamble. The module that selects the MAC PDUs to transmit is the *Scheduler*. Two schedulers are shipped with WiMsh: a simple first-in-first-out (FIFO) scheduler, which keeps the MAC PDUs into per-next-hop buffers and schedules them in the order of arrival; a Deficit Round Robin (DRR) [15] scheduler that stores data into per-traffic flow buffers. With DRR any traffic flow receives an amount of service that is proportional to its weight, as indicated by the WeightManager. With both FIFO and DRR, if the last Pdu served does not fit into the remaining space in the grant, it is fragmented: part of it is transmitted in the outgoing Burst, while the remaining bytes are kept into a *FragmentationBuffer*. The left over bytes will be transmitted to the neighbor in the subsequent grant. Fragmentation incurs the overhead of duplicating the MAC header, with the addition of a 1-byte fragmentation sub-header, for all fragments, but allows a more efficient usage of the negotiated slots, especially with large IP packets. The MAC PDUs received from the physical layer are passed through a *ReassemblyBuffer*, which returns a MAC SDU only when all its fragments have been correctly received. Due to channel errors, it is possible that only some of the fragments are received, in which case the SDU is lost.

3.3 Physical Layer

As already mentioned, in typical applications, the backbone of a WMN consists of fixed wireless routers. Therefore, links are established or torn down on a larger time scale than that of bandwidth scheduling and packet transmission. In WiMsh, we call *network topology* the logical view of the neighborhood relations between nodes. The network topology is configured by the user and it is stored as a directed graph into the *Topology* object, shared by all nodes, where the vertices represent nodes, and the edges links. The Topology is queried by the MAC and physical layer modules to determine which nodes belong to the first- and second-hop neighborhoods of any node, when needed. Another function

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

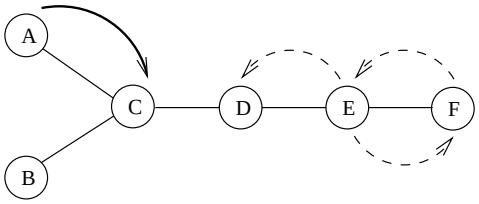

Figure 3: Protocol model example.

carried out by the Topology object is selecting the next-hop of packets, based on a path selection criterion defined in a class called *Forwarding*. Currently, paths are determined based on the shortest distance, in number of hops, between the source and destination node, where ties are broken randomly.

All the Burst objects created at the MAC, are passed to the physical layer (*Phy*) object, which models the nodes' radio transceiver. Data are then passed to one of the *Channel* objects, depending on which one was selected for transmission by the MAC. For each Burst received the Channel adds a propagation latency and then passes the Burst to the recipient Phy object. Control messages are broadcast, so they are passed to every node in the transmitting node's neighborhood. With both data and control messages, the Burst is not duplicated in memory for efficiency reasons, and it is destroyed by the Channel itself after dispatch. Therefore, the Phy module cannot modify the Burst content. The physical layer configuration is kept into the *PhyMib* object, which is shared among all nodes.

With regard to the wireless channel model, in the current version of the simulator we assume that simultaneous transmissions in the same slots are allowed only if the transmitting nodes lie at least two hops away from the receiver, which is known as "protocol model" assumption [13]. Specifically, a node is not considered to be able to decode transmissions from a neighbor node if either (i) the receiving node is transmitting on the same slot, or (ii) another neighbor of the receiving node is transmitting on the same slot. An example is illustrated in Fig. 3, where the transmission from node A to node C is assumed to be successful only if nodes B, C, and D remain silent, while transmissions from nodes E and F are allowed. In practice, there are several aspects of the wireless channel that cannot be captured by this model, such as transient phenomena (e.g. fast fading) and the cumulative effects due to the total number of simultaneous transmissions in the network. Nonetheless, we argue that the current assumptions of WiMsh are reasonable enough to carry out simulation studies that focus on the properties of the MAC layer and above. This statement is supported by the discussion in [5], where the authors prove that interference coming from second-hop neighbors can be considered to be negligible, at least in the environments with fixed nodes using a Time Division Multiple Access (TDMA) MAC protocol.

Optionally, a simple stochastic model can be used to enable imperfect data transmission. Specifically, a MAC PDU is marked as erroneous based on the value drawn from a uniform random variable. Different error rates can be set by the user for data and control, respectively. Furthermore, only for control messages, a channel error model inspired by [16] can be used. The model is based on a two-state discrete Markov chain, where the two states, namely good and bad, indicate whether the control message is successful received or not, respectively. With this model, transmissions are successful (unsuccessful) for a number of consecutive control slots drawn from a geometric distribution with average γ (β), hence the error rate is $e = \beta/(\gamma + \beta)$. State transitions of different links are not correlated to each other.

More accurate channel error models, e.g. taking into account the Signal-to-Interference-plus-Noise Ratio (SINR) experienced by the receiver, can be implemented by extending the Phy and Channel classes, while alternative path selection or forwarding functions can be implemented in more specialized versions of the Forwarding class. We note that the propagation models included in the ns-2 simulator, which are mostly intended for modeling networks of IEEE 802.11 devices, cannot be used without modifications with WiMsh, because they are designed to work with ns-2 packets, not Bursts. As already introduced in Section 3.1, this data structure is of paramount importance for modeling correctly the IEEE 802.16 MAC features of packing/fragmentation, which are not available for legacy ns-2 packets.

4. SIMULATOR CONFIGURATION

When performing network simulations, there are two main tasks that require a significant amount of effort: configuring the simulation scenario, and collecting/analyzing the output data.

The simulation scenario configuration problem, in turn, has two main causes. First, ns-2 requires the scenario to be written in the Tool Command Language (Tcl), which is an open source programming language born in 1990. Tcl is mostly used to produce dynamic HTML pages and in combination with the Tk toolkit to design graphical user interfaces. Knowledge of Tcl is not widespread today, because it has been supplanted by new programming languages specifically designed for such uses, like Java or Python. Therefore, it is very likely that the average ns-2 user is required to acquire at least some basic knowledge of Tcl before being even able to understand or modify the simple examples that can be found in tutorials and mailing lists. Second, ns-2 has been developed within and for the research community. Therefore, almost no effort has been committed on hiding the ns-2 internals or simplifying the scenario definition. The result is that one needs to learn the implementation details of the ns-2 core modules (queues, classifiers, timers, agents, ...) to produce a working scenario. This task is further complicated by the fact that: (i) the documentation is sparse and often obsolete; and, (ii) there are few or no consistency checks on the scenario definition, hence it is very difficult to spot errors without solid experience.

The second issue is that ns-2 offers very limited support to data collection and analysis. The most common way to analyze the output data is parsing the packet file traces, which are text files where each line reports the passing of one packet through some module. While some tools exist to carry out this process in an automatic manner, e.g. Tracegraph[10], this operation is not impractical for large simulation campaigns, due to the computational space and time overhead. To solve this problem in ns-2, one can use the *ns2measure* module [12], which collects performance measures during the simulation and saves them into binary files.

[10]http://www.tracegraph.com/

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

Parameter	Definition
run	replication identifier, use different values to obtain independent replications
duration	simulation duration, in seconds, including warm-up
warm	warm-up duration, in seconds
out	name of the output file containing measures
debug	name of the input file containing the functions to trace, leave blank to disable
startdebug	time when to start tracing functions

Table 3: Environment configuration.

These files can be post-processed using a set of publicly available tools [11] to produce both human-readable reports and ready-to-plot ASCII files. The module also computes the *t*-Student confidence intervals of the measures obtained from independent replications, which are useful for estimating the statistical confidence of the results. In WiMsh we used extensively the capabilities of ns2measure, which is included in the WiMsh's patch to ns-2. Several performance measures have been defined at application, MAC and physical layers, with different granularity: per packet, per traffic flow, per node, and network wide. The full list of the metrics, along with their brief description, is included in the WiMsh's website [12].

With regard to the scenario definition problem, we solved it by defining a specialized configuration file for IEEE 802.16 WMNs. This configuration file is parsed through a set of Tcl functions, which create the network and the traffic flows. This way, the user can run a wide variety of scenarios with zero knowledge of both Tcl and the ns-2 internals, by writing her own configuration file, as described in the following. The experienced user is still given the opportunity to change arbitrarily the scenario configuration, by modifying the Tcl source code[13] for parsing the configuration file.

The configuration file consists of lines of two types:

set opt(par) value

or

set opt(arr) { v1 v2 ... vN }

Statements of the first form are for defining scalar parameters (*opt* in the example), while those of the second form are for arrays (*arr* in the example). An alternative way of specifying the value of a parameter, either scalar or array, is to use command-line options. If the configuration file is called *test.tcl* and the ns-2 executable is in the default path, then the parameters above can also be specified by running:

ns test.tcl -par value -arr "v1 v2 ... vN"

The comprehensive list of the simulation parameters can be found in Table 3, while Table 4 contains the MAC layer parameters[14]. When a parameter is left unspecified, a default value is taken, which is often the correct choice. In the following we describe in detail how to set up the network topology and traffic configuration.

[11] http://cng1.iet.unipi.it/wiki/index.php/Ns2measure

[12] http://cng1.iet.unipi.it/wiki/index.php/Ns2mesh80216_Metrics_description

[13] The Tcl files can be found in the *wimax/tcl/* directory, and are: *header.msh*, *traffic.msh*, and *metrics.msh*

[14] Many parameters at the MAC layer are FEBA-specific. The interested reader can find a detailed description of FEBA in [9, 10]

Parameter	Definition
prio-weight	array of weights, one entry for each priority level
availabilities	advertise availability IEs in MSH-DSCH messages, when *on*
regrant	enable "re-granting", when *on*
request-fairness	be fair while requesting, when *on*
grant-fairness	be fair while granting, when *on*
regrant-fairness	be fair while "re-granting", when *on*
bwm-round-duration	bandwidth manager round duration F_{RR}, in bytes
weight-timeout	timeout to consider a flow inactive, in seconds
max-deficit	maximum deficit allowed per traffic flow, in bytes, 0 means infinite
max-backlog	backlog to consider a traffic flow unsatisfied, in bytes, 0 means infinite
weight-flow	enable traffic flow weighting, when *on*
grant-rnd-channel	select randomly the channel when granting slots, when *on*
dd-timeout	timeout to determine deadlock situations, in MSH-DSCH opportunities
min-grant	minimum size of a grant, in slots
hest-past	smoothing factor α to estimate the MSH-DSCH interval
hest-curr	$1 - \alpha$
scheduler	one of DRR (*fair-rr*), FIFO (*fifo*)
buffer	buffer size, in bytes
sch-round-duration	target DRR round duration F_{DRR}, in bytes
buffer-sharing	one of *per-flow* and *per-node*, only available with DRR

Table 4: MAC configuration.

The network topology can be selected among the set of pre-configured topologies illustrated in Fig. 4. These topologies are commonly employed in the performance evaluation of WMNs. Most of them are defined only in terms of a single parameter, called n, which has different meaning depending on the topology type. For instance, in a *chain* topology n is the number of nodes in the network, whereas in a *grid* it is the number of nodes per edge. Two topologies only, i.e. *multiring* and *star*, need another parameter, called *branches*, to be characterized. The meaning of n of *branches* for the different topologies can be inferred straightforwardly from Fig. 4. The full list of the physical layer parameters is reported in Table 5.

In addition to the topology, the MCSs employed for transmission in all the uni-directional links have to be selected. In the configuration file, the MCSs are identified via their index in Table 2. For each link, a new entry has to be added to three arrays, i.e. *prfsrc*, *prfdst*, and *prfndx*, to specify the source node, the destination node, and the MCS index, respectively. A default value is specified in *prfall* for all links that are not enumerated. For instance, the network in Fig. 5 can be specified by adding the following lines to the configuration file:

set opt(prfall) 0
set opt(prfsrc) { 0 1 2 }
set opt(prfdst) { 1 0 1 }
set opt(prfndx) { 2 1 4 }

We conclude with traffic flow set-up, which is similar to

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

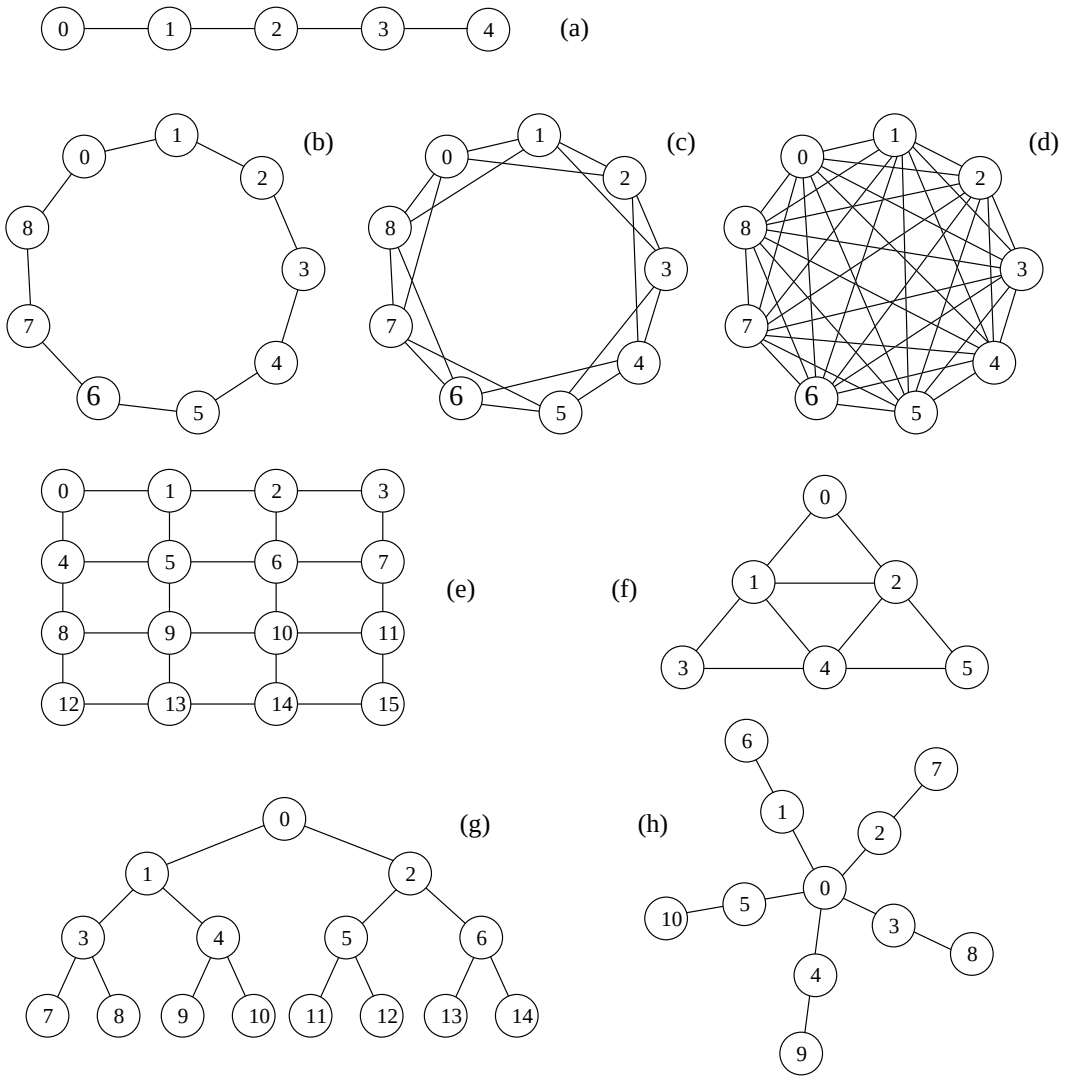

Figure 4: Pre-configured WiMsh topologies:(a) chain, $n = 5$; (b) ring, $n = 9$; (c) multiring, $n = 9$, $branches = 4$; (d) clique, $n = 9$; (e) grid, $n = 4$; (f) triangular, $n = 3$; (g) bintree, $n = 4$; (h) star, $n = 3$, $branches = 5$.

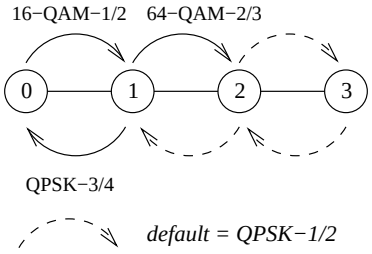

Figure 5: Example chain topology with four nodes.

the configuration of the links' MCSs. A traffic flow is characterized by seven values: the sender node, the destination node, the traffic type, the application start time, the application stop time, the number of aggregated sources per traffic flow, and the priority level. For each traffic flow, one entry has to be added to all seven arrays, called *trfsrc*, *trfdst*, *trftype*, *trfstart*, *trfstop*, *trfnsrc*, *trfprio*, respectively. A default value exists for all parameters, except the source and destination nodes. Five types of applications are pre-configured: Voice over IP (*voip*)[15], Constant Bit-Rate (*cbr*), Video on Demand (*vod*), super-imposition of four independent Poisson processes [4] (*bwa*), and uninterrupted source (*ftp*). All the applications use User Datagram Protocol (UDP) agents, except FTP that uses Transport Control Protocol (TCP). For instance, with the following lines:

set opt(trfsrc) { *0 3 2* }
set opt(trfdst) { *3 0 3* }
set opt(trftype) { *voip voip cbr* }

[15]VoIP traffic is managed by the *ns2voip* module, also available as a stand-alone ns-2 add-on at http://cng1.iet.unipi.it/wiki/index.php/Ns2voip.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

Parameter	Definition
topology	see Fig. 4
n	see Fig. 4
branches	see Fig. 4
channel	number of channels
propagation	physical propagation latency, in μs
bandwidth	channel bandwidth, in MHz (see Table 1)
cyclic-prefix	one of 1/4, 1/8, 1/16, 1/32
sym-duration	OFDM symbol duration, in μs, ignored if *bandwidth* is specified
sym-perframe	number of OFDM symbols per frame, ignored if *bandwidth* is specified
control	number of control slots per frame, between 2 and 16
cfg-interval	interval between consecutive control frames used for network configuration, in frames
holdoff-dsch-def	default XmtHoldoffExponent used for MSH-DSCH messages
holdoff-dsch	array of XmtHoldoffExponent values, one per node, used for MSH-DSCH messages
holdoff-ncfg-def	default XmtHoldoffExponent used for MSH-NCFG messages
holdoff-ncfg	array of XmtHoldoffExponent values, one per node, used for MSH-NCFG messages
max-advertised	maximum number of neighbors advertised in MSH-DSCH messages, -1 means infinite
chan-data-per	uniform MAC PDU error probability (data)
chan-ctrl-per	uniform MAC PDU error probability (control)
msh-dsch-avg-bad	number of consecutive corrupted MSH-DSCH messages, on the average
msh-dsch-avg-good	number of consecutive uncorrupted MSH-DSCH messages, on the average

Table 5: Physical layer configuration.

set opt(trfprio) { 1 1 0 }
set opt(trfstart-def) 2.0
set opt(trfstop-def) 10.0
set opt(trfnsrc-def) 1

a bi-directional VoIP conversation is established between node 0 and node 3, with higher priority than the uni-directional CBR traffic flow from node 2 to node 3. All the traffic flows start at time 2 s and stop at time 10 s, and carry data from a single application. The traffic configuration parameters are summarized in Table 6.

4.1 Educational Use

Network simulators often play an important role as educational tools in under-graduate courses. In fact, with the help of a network simulator, the student can experiment in a controlled software environment the network applications and protocols studied. Carrying out the same trials with real equipment is much more time-consuming, for both students and teachers. Also, the set-up cost of a simulation laboratory is virtually zero, provided that a free-of-charge simulation tool is employed, because any general-purpose computer room can be used for this purpose.

However, we argue that some features of the simulation tool are required to actually stimulate the student's creative and learning side. First, the tool must be *easy* to use without special training. Unfortunately, many open source simulators are not user-friendly. For instance, spending serious time learning a new scripting language, like Tcl for ns-2, might be reasonable if you are a graduate student or a research associate, planning to use the tool in the next years for your research. However, this is an obstacle if the objective is to to run a small set of toy experiments, e.g. as part of a project for an undergraduate course. Second, the simulator should allow the user to run narrow-scope simulations. Given the limited time (and, sometimes, inclination), students might not be keen on reading the whole documentation of the simulator for tuning accurately all the network and system parameters required for an experiment. It is likely that they would end up using the wrong, possibly inconsistent, set of inputs, possibly hiding the most interesting phenomena. Sometimes, they do not even have the technical background required to configure everything correctly. Finally, the simulation results should be ready for analysis. If the simulator itself does not have tools for visualizing the output data, e.g. by means of tables or plots, the students might find it difficult to walk through the raw data produced by the simulations, which might easily overwhelm them.

Parameter	Definition
voip-model	Voice Activity Detection (VAD) model, one of *one-to-one*, *one-to-many*, *many-to-one*, *many-to-many*
voip-codec	VoIP codec, e.g. *GSM.AMR*, see [3]
voip-aggr	number of speech samples per VoIP frame
bwa-rate	rate of *bwa* traffic specified by [4], in b/s
bwa-pkt	packet size of *bwa* traffic specified by [4], in bytes
cbr-rate	rate of CBR traffic, in b/s
cbr-pkt	packet size of CBR traffic, in bytes
cbr-rnd	set to 0 to have perfect CBR generation
vod-trace	ns-2 trace file of an encoded VoD stream
ftp-wnd	maximum TCP window for FTP traffic
ftp-pkt	TCP segment size for FTP traffic

Table 6: Traffic configuration.

We have used ns-2 as an educational tool in the course of Advanced Networking Architectures and Wireless Systems, at the Faculty of Engineering of the University of Pisa, since 2003. Based on our experience, we claim that WiMsh complies with all the requirements discussed above. In fact, its configuration file totally hides the complexity of writing the scenario configuration in Tcl, and provides a straightforward way of experimenting with pre-configured sets of traffic types and network topologies. At the same time, data can be visualized as 2-d plots using the *ns2measure* module. Finally, as both ns-2 and WiMsh are distributed under GPL, and they run under most operating systems, WiMsh can be used without limitations both at school and at home.

5. CONCLUSIONS

In this paper we have described the WiMsh module, which can be used to simulate multi-channel WMNs using IEEE 802.16 mesh with distributed scheduling in ns-2. The simu-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679

lator has been made publicly available under GPL in October 2007, and has been downloaded 2000+ times since then. The core functions at the MAC layer are the following. First, the Coordinator implements the distributed election procedure, which is used to access in a collision-free manner the slots in the control sub-frame. Second, the BwManager is used to negotiate access to the slots in the data sub-frame in a distributed manner. This is done via the FEBA algorithm, which provides traffic flows with weighted fair access to the network resources. Finally, packet scheduling is done by the Scheduler, which implements FIFO and DRR algorithms.

A specific design goal of WiMsh is to provide the user with a simple, yet powerful, interface for configuring the simulation scenario. This is achieved by means of a configuration file that does not require the user to understand the ns-2 internals or to learn the Tcl scripting language, as expected to run simulations in "plain" ns-2. Therefore, the novice users can experiment with WiMsh without the need of special training or broad simulation and networking background, thus focusing on the specific aspects of the IEEE 802.16 mesh simulations they are running. This makes WiMsh suitable for use as an educational tool in networking courses.

6. REFERENCES

[1] Air Interface for Fixed Broadband Wireless Access Systems. *IEEE 802.16d*, 2004.

[2] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: A survey. *Computer Networks (Elsevier)*, 47:445–487, Mar. 2005.

[3] A. Bacioccola, C. Cicconetti, and G. Stea. User-level performance evaluation of voip using ns-2. In *ICST NSTools*, pages 1–10, 2007.

[4] C. R. Baugh, J. Huang, R. Schwartz, and D. Trinkwon. Traffic model for 802.16 TG3 MAC/PHY simulations. Technical report, IEEE 802.16 Broadband Wireless Access Working Group, Mar. 2001.

[5] G. Brar, D. M. Blough, and P. Santi. Computationally efficient scheduling with the physical interference model for throughput improvement in wireless mesh networks. In *ACM MobiCom*, pages 2–13, 2006.

[6] R. Bruno, M. Conti, and E. Gregori. Mesh networks: commodity multihop ad hoc networks. *Communications Magazine, IEEE*, 43(3):123–131, March 2005.

[7] M. Cao, W. Ma, Q. Zhang, and X. Wang. Analysis of IEEE 802.16 mesh mode scheduler performance. *IEEE Trans. Wireless Commun.*, 6(4):1455–1464, Apr. 2007.

[8] M. Cao, V. Raghunathan, and P. Kumar. A tractable algorithm for fair and efficient uplink scheduling of multi-hop WiMAX mesh networks. In *IEEE WiMesh*, pages 93–100, 2006.

[9] C. Cicconetti, I. F. Akyildiz, and L. Lenzini. Bandwidth balancing in multi-channel IEEE 802.16 Wireless Mesh Networks. In *Proc. IEEE Infocom*, Anchorage (USA), May 2007.

[10] C. Cicconetti, I. F. Akyildiz, and L. Lenzini. FEBA: A bandwidth allocation algorithm for service differentiation in IEEE 802.16 mesh networks. *IEEE/ACM Trans. Networking*, Oct. 2009.

[11] C. Cicconetti, A. Erta, L. Lenzini, and E. Mingozzi. Performance evaluation of the mesh election procedure of IEEE 802.16/WiMAX. In *Proc. IEEE MSWiM*, pages 323–327, Chania, Crete Island, Greece, 22-26 April 2007.

[12] C. Cicconetti, E. Mingozzi, and G. Stea. An integrated framework for enabling effective data collection and statistical analysis with ns-2. In *Proc. WNS2*, Pisa, Italy, Oct. 2006.

[13] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Trans. Inform. Theory*, 46:388–404, Mar. 2000.

[14] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. ns-3 project goals. In *ICST WNS2*, page 13, 2006.

[15] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/ACM Trans. Netw.*, 4(3):375–385, 1996.

[16] W. Turin. *Performance analysis of digital transmission systems*. Computer Science Press, 2000.

[17] S. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, C. Chiou, and C. Lin. The design and implementation of the NCTUns 1.0 network simulator. *Compute Networks (Elsevier)*, 42(2):175–197, June 2003.

[18] X. Wang and A. O. Lim. IEEE 802.11s wireless mesh networks: Framework and challenges. *Ad Hoc Netw.*, 6(6):970–984, 2008.

APPENDIX

A. CHANGES TO NS-2

The WiMsh module is provided as a patch to either ns-2.31 or ns-2.33. The patch includes also other modules, which have been developed by the Computer Networking Group of the University of Pisa[16]. The installation instructions can be found on WiMsh's website[17]. After applying the patch, ns-2 is modified as follows.

A new directoy is created, called *wimax* in the root ns-2.3*x* directory. The C++ files are included in the directories *wimax/mesh* and *wimax/common*, while the Tcl library files are in *wimax/tcl*. The latter also contains an example, called *mesh.tcl*, which can be executed to verify that the simulator has been patched correctly. This example file is well commented and can be used as the starting point for configuring the planned simulations. Note that, since the *Makefile.in* file is modified, so as to include the new C++ object files in the compilation and linking phases, it is necessary to re-run the *configure* script, already provided in the original ns-2 distribution. If object files have been already compiled before patching, they must be removed before compilation to ensure consistency, which can be done with the command *make clean*.

In addition to *Makefile.in*, the only files of the original ns-2 distribution that have ben modified are those for connecting the Link Layer to the IEEE 802.16 MAC objects, as illustrated in Fig. 2, that is: *mac/ll.h*, *mac/ll.cc*, *tcl/lib/ns-lib.tcl*, and *tcl/lib/ns-mobilenode.tcl*.

[16] http://cng1.iet.unipi.it/wiki/index.php/Software
[17] http://cng1.iet.unipi.it/wiki/index.php/Ns2mesh80216

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5679
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5679