An IEEE 802.11g simulation model with extended debug capabilities

Sorin Cocorada
University Transilvania of Brasov
Politehnicii 1
500024 Brasov-Romania
+40268478705
sorin.cocorada@unitbv.ro

ABSTRACT
IEEE 802.11g has become the de facto standard for Wireless LAN. Most of the 802.11g functionality is implemented in hardware or firmware; modifying or extending such communication protocols is a difficult task. In this paper, we present modifications to the Omnet++ simulation environment to support an accurate IEEE 802.11g MAC and PHY simulation model. A solution for debugging the proposed model using protocol analyzers is also provided.

Categories and Subject Descriptors
I.6 [**Simulation and Modeling**]: Wireless Simulation

General Terms
Design

Keywords
IEEE 802.11g, Simulator, Omnet++, Wireshark

1. INTRODUCTION
Wireless local area networks (WLANs) have become an important component of the Internet infrastructure. IEEE 802.11g [4] defines the physical layer and medium access control layer for WLANs as the de facto standard for such networks.

Communication protocols running at the data link layer are mostly implemented in hardware or firmware because of the severe timing requirements (amounts of micro-seconds). Under these circumstances, modifying or extending existing protocols may be difficult or even impossible.

In order to experiment, modify and extend the IEEE 802.11g standard, a simulation model of the standard is needed that allows the efficient evaluation of the system performance. Such a model must be extensible, modular, easy to understand, and fast.
Until now, several network and protocol simulators have been released, both open source and commercial ones, but not all of them support IEEE 802.11g. For example NCTUns [6] provides a good graphical interface but supports only 802.11b; Pythagor [8] supports 802.11a/b/g but it is limited to data link layer simulations and it is not possible to simulate additional protocols running at higher layers.

Commercial simulators (like Opnet) are focused especially on designing and optimizing networks by tuning the parameters of the existing protocols and less on developing new protocols. Ns2 [9] also supports 802.11a/b/g but Omnet++ is more flexible, provides a better encapsulation for protocols and also includes a graphical user interface (GUI) for debugging, tracing or presenting simulations.[7].

The paper is structured as follows. In Section 2, our simulation model is described including physical and data link layer modeling, Section 3 describes a simple method for debugging the proposed model in real time using network protocol analyzers. Some concluding remarks are provided in the final section.

2. OMNeT++ SIMULATION ENVIRONMENT DESCRIPTION
Many communication systems are often simulated using *discrete event simulation (*for example with the Omnet++ simulator [7] or ns2 simulator [9]). It works as follows: the core of the method is to use a global time *currentTime* and an *event scheduler*. Events are objects that represent different transitions; all the events have an associated firing time. The event scheduler represents a list of events, sorted by increasing firing times. The simulation kernel selects the first event in the event scheduler, advances currentTime to the firing time of this event, and executes the event. The execution of an event may schedule new events with firing times greater or equal to currentTime, and may change or delete events that were previously listed in the event scheduler. The global simulation time currentTime cannot be modified by an event. Thus, the simulation time jumps from one event firing time to the next – hence the name of discrete event simulation. In addition to simulating the logic of the system which is being modeled, events have to update different statistics counters.

In order to provide an IEEE 802.11g model, we extended the simulation environment described in [7]. We have modified INET Framework 20061020 (running above Omnet++ v3.4b2) PHY model and MAC layer parameters from 802.11b to 802.11g

OMNET++ 2008, March 03, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.3022

standard specification [4]. These parameters include MAC and physical layer convergence procedure (PLCP) header formats, data rates and use of forward error correction (FEC) [2].

2.1 Physical layer modeling

We have modified this simulator in order to consider the effect of the wireless physical layer in modeling WLANs. Physical layer parameters like path loss, fading, interference and noise have been taken into account because of their important effects in simulation results. The simulation uses Friis propagation model with a path loss exponent which can be configured at run time. To calculate the bit error rate (BER) at the receiver, an additive white Gaussian noise (AWGN) channel has been used and then a Rayleigh fading channel. To produce more realistic results it is recommended to back off the transmitter output power for higher OFDM data rates to reduce the impact caused by the power amplifier non-linear distortion. During the simulation, a constant noise level is assumed. The thermal noise level for a 20 MHz channel is approximately -101.7 dBm and up to 5 dBm noise from the amplifier chain can be included [5].

Table 1. IEEE 802.11g PHY modes

Mode	Modulation	Coderate	Bitrate
1	BPSK	1/2	6 Mbps
2	BPSK	3/4	9 Mbps
3	QPSK	1/2	12 Mbps
4	QPSK	3/4	18 Mbps
5	16-QAM	1/2	24 Mbps
6	16-QAM	3/4	36 Mbps
7	64-QAM	2/3	48 Mbps
8	64-QAM	3/4	54 Mbps

For a white Guassian noise over the wireless medium, the bit error probability (Pb) depends on the modulation scheme employed. For a 2k-ary QAM modulation with Gray coding and k=2, 4, 6 the approximate bit error probability is calculated using a recursive algorithm provided in [3], according to:

$$P_b^{(2^k)}(\gamma) = \frac{2^{2-\frac{k}{2}}}{k}\left(Q\left(\sqrt{\gamma}\right) + Q\left(3\sqrt{\gamma}\right)\right) + \frac{k-2}{k}\cdot P_b^{2^{k-2}}\left(\gamma\right) \quad (1)$$

where k represents the number of bits per symbol of a 2^k-QAM constellation, $P_b^{(4)} = Q(\sqrt{\gamma})$ and $\gamma = 3k/(2^k-1)\cdot SNIR\cdot (Bandwidth/BitRate)$.

In case of a Rayleigh fading channel, we used the following formulas deduced from [1]:

$$\begin{aligned} P_b^{(4)} &= \frac{1}{2}\left(1-\sqrt{\frac{\gamma}{1+\gamma}}\right) \\ P_b^{(16)} &= \frac{5}{8} - \frac{3}{8}\sqrt{\frac{2\gamma}{5+2\gamma}} - \frac{1}{4}\sqrt{\frac{18\gamma}{5+18\gamma}} \\ P_b^{(64)} &= \frac{13}{24} - \frac{7}{24}\sqrt{\frac{\gamma}{7+\gamma}} - \frac{1}{4}\sqrt{\frac{9\gamma}{7+9\gamma}} \end{aligned} \quad (2)$$

where γ represents the average signal to noise ratio per bit. For 802.11g, FEC Viterbi decoding is assumed in the receiver side. We have used the upper bond probability of error that is given in [1] under the assumption of binary convolutional coding and hard decision (HDD) Viterbi decoding. Specifically, for a packet of L bytes this probability is:

$$P_p(L) = 1-(1-P_b)^{8L} \quad (3)$$

Then we can upper bound the bit error probability as:

$$P_b \le \sum_{d=d_{free}}^{\infty} a_d P_d \quad (4)$$

where d_{free} is the free distance of the convolutional code, a_d is the total number of error events of weight d and P_d is the probability that an incorrect path at distance d from the correct path is chosen by the Viterbi decoder. Note that (4) is valid for both HDD and soft decision (SDD); however, P_d is different for HDD and SDD. The a_d coefficients are code dependent but it is generally accepted that the first five terms in (4) are dominant. For HDD, the algorithm uses Hamming distance as the metric. When d is odd, the probability of selecting the incorrect path is:

$$P_d = \sum_{i=(d+1)/2}^{d} C_i^d p^i (1-p)^{d-i} \quad (5)$$

where p is the probability of channel bit error. When d is even, the probability that an incorrect path at distance d from the correct path should be chosen by the Viterbi decoder is:

$$P_d = 0.5 C_{d/2}^d p^{d/2}(1-p)^{d/2} + \sum_{i=d/2+1}^{d} C_i^d p^i (1-p)^{d-i} \quad (6)$$

To improve the execution time, a lookup table has been used for calculating the above combinations.

2.2 Data link layer modeling

To model the data link layer, the existing 802.11b finite state machine (FSM) has been reused with few modifications to reflect the 802.11g parameters. Slot time has been reduced from 20μs to 9μs but it can take any other value. The duration field of the frames transmitted at 802.11g rates has been calculated using the formula provided in the standard [4]. A small bug in the original 802.11b FSM has been fixed because, according to the 802.11 specifications, stations must back off after transmitting a frame. The parameters which have been added or extended to the

existing 802.11b model to control the proposed 802.11g model and their possible values are listed in Table 2.

Table 2. Simulation parameters

Parameters	Value
mac.opMode	'g' for 802.11g , 'b' for 802.11b
mac.slotTime	9us for 802.11g-only
mac.bitrate	6e6, 9e6, 12e6, 18e6, 24e6, 36e6, 48e6, 54e6
mac.AIFSN	2 for DIFS
radio.phyOpMode	'g' for 802.11g , 'b' for 802.11b
radio.channelModel	'a' for AWGN, 'r' for Rayleigh
radio.bitrate	6e6, 9e6, 12e6, 18e6, 24e6, 36e6, 48e6, 54e6

3. Debugging the model using Wireshark

Debugging protocols in Omnet++ is done through the graphic user interface in which packet fields can be inspected as they are processed. Normally, the protocol analysis is done by means of dedicated network protocol analyzers such as Wireshark. To achieve this it is necessary that the packet headers that are represented in Omnet++ as C++ objects should be first converted in network byte order using header serializers (Figure 1). In this approach AirFrame objects are serialized to radiotap headers [5] using the following fields: MAC timestamp (simulation time), rate (data rate of the transmitted frame), channel and transmit power (in dBm), which are relevant for our simulation. A Ieee80211Frame serializer has also been added. For the other headers (IPv4, ARP etc) the existing serializers have been employed. Logical link control (LLC) headers are added only to Ieee80211 frames which carry data from upper layer protocols. Serialized packets can be written to a real or virtual network interface (using libpcap or raw sockets) which supports the desired encapsulation and from here they can be captured in real time using Wireshark or another similar tool. The model has been tested with a madwifi version 0.9.3.3 [5] interface in monitor mode and with the radio transmitter disabled.

4. CONCLUSION

In this paper, a complete IEEE 802.11g simulation model was presented along with a real time debugging solution using network protocol analyzers. The performance of 802.11 transmissions is highly dependent upon the wireless channel model, transmitter output power, modulations and data rates which have been considered in this model.

In future works, we are going to investigate the possibility of using Omnet++ for transmitting IEEE 802.11 frames over the air and to create a software access point which can be used for experimenting, modifying and extending the IEEE 802.11 standard features.

5. ACKNOWLEDGMENTS

This work was supported by CNCSIS Romania under contract 90/01.10.2007 (TD_156).

6. REFERENCES

[1] Goldsmith, A. 2005. Wireless Communications. Cambridge University Press.

[2] Hossein, M., Turletty, T. 2003. Simulation-Based Performance Analysis of 802.11a Wireless LAN. Proceedings of the Int'l Symposium on Telecommunications (Isfahan, Iran, August 16-18, 2003), 758–762.

[3] Yang, L. , Hanzo, L. 2000. A Recursive Algorithm for the Error Probability Evaluation of M-QAM, IEEE Communications Letters, Vol. 4, No. 10, (Oct. 2000)

[4] IEEE 802.11g, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band, http://grouper.ieee.org/groups/802/11/

[5] Multiband Atheros Driver for WiFi http://www.madwifi.org

[6] NCTUns network simulator and emulator http://nsl10.csie.nctu.edu.tw/

[7] Omnet++ Simulator, http://www.omnetpp.org/

[8] Pythagor simulator http://www.icsd.aegean.gr/telecom/Pythagor/

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

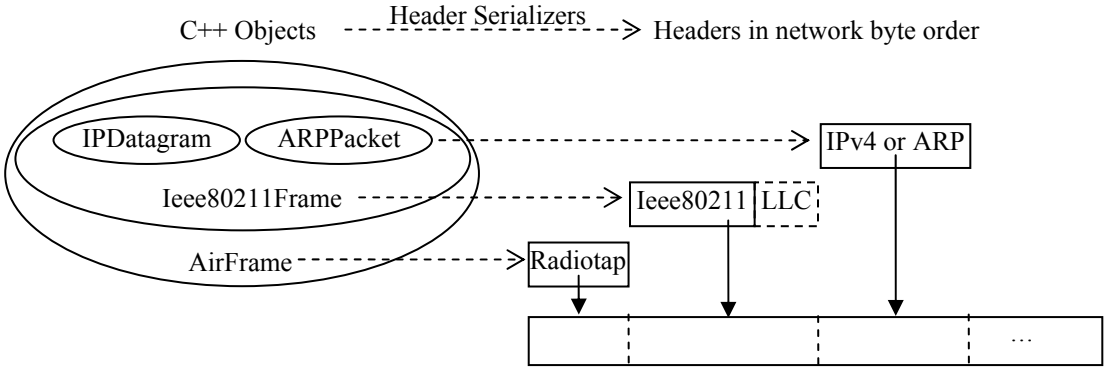

Figure 1. Converting between object headers and network byte order headers