Novo módulo IEEE 802.11 para o NS 2

New IEEE 802.11 MAC module for NS 2

Eduardo Henrique Molina da Cruz[1]; Elvio João Leonardo[2]

Resumo

Este artigo apresenta uma nova implementação para o protocolo de controle de acesso ao meio do IEEE 802.11 para ser utilizado no simulador NS 2. É de conhecimento geral que a implementação corrente desse protocolo no NS 2 é semanticamente e sintaticamente incorreto. Esse fato motivou-nos a projetar e implementar uma alternativa que fosse mais próxima do protocolo especificado, e, além disso, oferecer algo de entendimento e manutenção mais fácil. O módulo proposto está baseado em máquinas de estados finitos, onde as relações entre eventos e procedimentos são facilmente identificáveis. As mudanças propostas foram inseridas no NS 2 e verificadas em comparação à implementação original. O desempenho do novo módulo mostrou-se bom, sendo que as diferenças de comportamento encontradas devem-se às falhas existentes na implementação atual.
Palavras-chave: MAC. Simulador NS 2. IEEE 802.11.

Abstract

This article presents a new implementation of the IEEE 802.11 Medium Access Control (MAC) protocol to be used with the network simulation tool NS 2. It is well known that the current implementation of this protocol is faulty both semantically and syntactically. This fact has motivated us to design and implement an alternative that is closer to the protocol specified by the standards, and that it is easier to understand and to change. The proposed module is based on a few finite state machines, with links between events and procedures easily understood. The proposed changes were added to NS 2 and tested against the original implementation. The new module performed well and the differences found can be justified by the faults present in the original implementation.
Key words: MAC. Network Simulator 2. IEEE 802.11.

[1] Undergraduate student, Computer Science, State University of Maringa; E-mail: eduardohmdacruz@gmail.com.
[2] Assistant Professor, Informatics Department, State University of Maringa, E-mail: elvio.leonardo@din.uem.br.

Introduction

Network simulation is commonly used to obtain information about the behavior of networks prior to its deployment, mimicking an environment as close as possible to the real one. The information acquired can be used in several ways: to help design the network, to find its optimum parameters and topology, or its bottlenecks, to evaluate new ideas and test new solutions, as well as academic purpose.

The Network Simulator 2 (FALL, 1999) is a powerful toolkit for simulating networks. Unfortunately, its implementation of the MAC 802.11 protocol disappoints in many ways. The results it gives are not necessarily correct (HENDERSON et al., 2006; SCHMIDT-EISENLOHR; LETAMENDIA-MURUA; TORRENT-MORENO, 2006; CHEN, 2007) because it has semantic issues, the code is badly written and, although it is coded in C++, it does not follow the idea of Object Orientation (OO).

In our solution, we have focused in the IEEE specification, following it whenever it was possible. By taking advantage of OO, we have programmed each finite state machine that composes the MAC Layer separately. By doing this we have achieved clean and compact software module for the IEEE 802.11.

This article is organized as follows: Section 2 presents an overview of the IEEE 802.11 protocol. Section 3 discusses the problems of the current implementation of the protocol, and Section 4 presents the new implementation. A conclusion is given in Section 5.

Overview of 802.11 MAC protocol

Wireless networks are much more flexible than the traditional wired ones. They allow networks to be deployed without the hard work and the expensive costs associated to the installation of wires and building modifications. Besides, wireless networks allow communication at places that would not be practical with wired ones, such as airports, restaurants and other public areas.

The IEEE 802.11 standards (CROW, 1997; IEEE, 1999) support both ad-hoc and infra-structured network configurations. In ad-hoc networks, stations can directly connect to another one. On the other hand, in infra-structured networks, a usually fixed point, called AP (Access Point) works as a base station and centralizes all communication between terminals.

The BSS (Basic Service Set) comprehends a group of stations that are under control of a coordination function, and it is the base of the IEEE 802.11 architecture. The protocol specification covers functions in several layers, such as the Physical Layer and the MAC Sublayer. The focus of this study is with the MAC Sublayer.

There are two modes of operation in a BSS: the DCF (Distributed Coordination Function) and the PCF (Point Coordination Function). The DCF is designed for asynchronous data transfers and it is able to guarantee the integrity of the delivered data. It is used in ad-hoc networks. PCF is a mode of operation suited to the quick transmission of delay sensitive data, such as media streaming. It is based on polling stations, with the right of transmission and the synchronism controlled and centralized in the AP. This work's concern is limited to the DCF mode since this is the only one available in current implementation of the network simulator.

The DCF is the standard access method for exchanging data. It operates alone in ad-hoc networks and together with the PCF in infra-structured ones. The DCF is based on a carrier sense with collision avoidance mechanism referred to as virtual carrier sense.

The MPDU (MAC Protocol Data Unit) contains in its header information about the packet, like the transmission duration and a 32-bit number (CRC) used to check data integrity. The duration field is used by stations to perform the virtual carrier sense,

updating their NAV (Network Allocate Vector) with the period of time that the environment will stay busy.

If the medium is busy at the start of a transfer, or it does not remain idle for at least DIFS (DCF Interframe Space), the station must wait the medium to become idle and then it should go to backoff state. Otherwise, transfers may begin. A hand-shaking is performed whenever the size of an MPDU exceeds a constant known as RTS Threshold. The RTS (Request To Send) is a small packet sent to test if the destination station can accept transfers. If it does, it sends back to the source station a CTS (Clear To Send) packet after a period known SIFS (Short Interframe Space). After this is done, the MPDU transfer may begin after another SIFS. If it completes successfully, the destination checks the integrity of received data by looking at the 32-bit CRC. If it is correct, it sends an ACK (Acknowledgment) to the source after a SIFS time. If any of these steps fails, the source station waits for an EIFS (Extended Interframe Space) period and then goes to backoff state. Please, note that broadcast packets do not have handshake and acknowledgment.

It is important to say that the backoff procedure must be called only if the medium stays idle for an IFS (Interframe Space) period. When a station goes to backoff state, it stays idle for a time randomly generated and then try to re-send the packet. The backoff timer is paused whenever the medium becomes busy. Note that, theoretically, all the other stations in the BSS hear the packets and set up their NAV, avoiding collisions if they receive a packet from the transmitting station.

```
Inline int Mac802_11::is_idle() {
    if(rx_state_ != MAC_IDLE)
       return 0;
    if(tx_state_ != MAC_IDLE)
        return 0;
    if(nav_ > Scheduler::instance().clock())
        return 1;
    return 1;
}
```

(a)

```
Void Mac802_11::backoffHandler() {
    if(pktCTRL_) {
        assert(mhSend_.busy()||mhDefer_.busy());
        return;
    }
    if(check_pktRTS() == 0)
        return;
    if(check_pktTx() == 0)
        return;
}
```

(b)

Figure 1. Code samples of current implementation: (a) Method that checks if the MAC is idle; (b) Method called when backoff timer expires.

Problems of the original implemen-tation

The NS version of the MAC 802.11 described here is version 2.29, so there might be some divergences for other releases.

In the original code, the class named Mac802_11 is the "kernel" of everything. All the states machines are programmed inside it, creating a disorganized environment. An explanation of the design can be found in (FALL, 2006; LIU, 2008). The code is hard to understand and in some cases it does not simulate the protocol as specified by IEEE, possibly generating incorrect results.

There are other classes, mostly of them just timer handle classes, of which we will talk about later. Here, we will present some sections of code and comment them, showing the importance in creating a replacement for the official implementation.

The Code. The class Mac802_11 mentioned earlier takes care of all the State Machines, controls the timers and sends and receives the packets; this

is not Object Orientation. According to (AMBLER, 2001), the concept of OO is to divide and conquer, programming separate different modules that, independently, solves small problems, but, together, solves a common and larger one.

There are two classes, PHY_MIB and MAC_MIB, which purpose of existence is a dilemma, because the only thing they do is to store and return values.

One of the concepts of a well written function is to avoid too many return statements, as well as to use more conditional clauses than it is actually needed. For instance, in Fig. 1a, the target test values are the same in the first two tests. Therefore, applying some Boolean test would improve the performance by transforming these two conditional statements into a single one. Besides, the three clauses lead to the same result, so only one if statement is really necessary, helping saving some processing time. In Fig. 1b, a 10-line procedure has 3 return statements, which should be avoided.

Timers. Timers are used during the simulation to measure the transfer and interframe spaces time intervals. The NS takes care of the low level programming, but it is MAC's responsibility to configure properly the duration of the timer and the callback procedure invoked when timer expires.

The official source implements a class named MacTimer that contains a big part of the logic. For each timer required by any event, the programmer should designate separated classes that inherit the MacTimer class. So, as the simulation has lots of different components with diverse purposes, there are a lot of timer classes, and, basically, the only thing that differs one from another is what they do when the timer expires.

Although this is an efficient mechanism, it is not the best solution. When a sector asks for a timer, someone who is reading the code has to search the appropriate class that handles the timer to check what is done when it expires. After this, the person must search the routine called by the handler. In other words, it is not an easy task to understand what exactly is happening and in fact it is too much effort for such a little procedure.

Semantic Bugs. Besides the below standard coding practice issues, there are some logic errors too.

When sending packets, as explained before, the environment must be idle for a DIFS period; otherwise the station must wait the transmission to finish and when the medium stays idle for an IFS period, backoff is started. In the original source code, when the medium is not idle and a packet is ready to be sent, it goes to backoff directly with the timer paused. This is wrong, and in a collision situation, it may cause fake results.

When receiving a packet, if it arrives with error, nothing can be done. However, the built-in code read its duration field to set up the NAV. How, in real conditions, the MAC could possibly read anything from corrupted data?

Something similar happens in some collision situations. When a packet is being received, and another one arrives from the lower layer, the protocol checks their power levels, and then decides which one should be received. In the analyzed code, when the packet already being received has greater signal strength, it might happen that the NAV gets set using the duration field of the weaker packet. The problem is that this packet has not arrived yet and, therefore, no data can actually be extracted from it.

All the mistakes and bad project design mentioned here has motivated us to develop a better simulation module, more reliable, easier to understand and more faithful to the IEEE specification.

```
class class_test {
      protected:
        int test;
      public:
        class_test() {
          this->test = 12345;
        }

        void method_test() {
          printf("test %i\n", this->test);
        }
    };

    class_test t;
    void (*callback)();

    callback = t.method_test;
    (*callback)();
```

(a)

```
    class class_test {
      protected:
        int test;
      public:
        class_test() {
          this->test = 12345;
        }

        static void method_test(class_test *thiss) {
          printf("test %i\n", thiss->test);
        }
    };

    class_test t;
    void (*callback)(class_test*);

    callback = &class_test::method_test;
    (*callback)(&t);
```

(b)

Figure 2. Use of member function pointers: (a) incorrect; (b) correct.

The Proposed Module

In the proposed code (CRUZ, 2008), we took the original flowchart presented in the ANSI/IEEE Standard 802.11 as the base design. The state machines described there are almost the same we use in our solution. The classes used to implement them are listed below.

- mac_802_11_backoff: the backoff state machine; it has the states given below.
  - NO_BACKOFF: When backoff is not being performed.
  - BACKOFF: When backoff is being performed.
  - PAUSED: When the backoff timer is paused due to medium (channel) activity. When the medium becomes idle again, the state machine returns to BACKOFF.
- mac_802_11_tx_coord: the tx_coordination machine. It handles the transmission of outgoing packets (RTS and MPDU). It is the most complex of the protocol's states machines and receives signals from the other machines when some events occur. See below for more details. It has the states given below.
  - IDLE: When no packet is being transmitted.
  - WAIT_RECVED: When a packet is in send procedure, but the MAC is receiving a packet from the Physical Layer.
  - WAIT_NAV: When a packet is in send procedure, but the MAC is halted due to NAV.
  - WAIT_IFS_BACKOFF: Interframe space before going for backoff.
  - WAIT_BACKOFF: Assumed when the backoff machine is not in the NO_ BACKOFF state.
  - WAIT_DIFS: DIFS interframe space before sending a packet.
  - WAIT_CTS: State assumed after the RTS was sent.
  - WAITSIFS_CTS: When the CTS is received, this state remains active for a SIFS period.
  - WAIT_ACK: State called after the MPDU is sent.
  - WAIT_TX: Assumed when a reception is under way (MAC is sending or will send a CTS/ACK) and a packet is requested to be sent (by LLC).
  - SENDING_MPDU: State that begins when an MPDU is being sent but no ACK is required (e.g., broadcast packets).
  - BACKOFF_TXDONE: Backoff that is performed after a transmission process ends.

```
class class_test {
      protected:
          int test;
      public:
          class_test() {
              this->test = 12345;
          }

          static void method_test(void *obj) {
              class_test *thiss = (class_test*) obj;
              printf("test %i\n", thiss->test);
          }
    };

    class general_timer {
      public:
        void (*callback)(void *);
        void *myowner;

        void finished_timer () {
            (*this->callback)(this->myowner);
        }
    };

    general_timer timer;
    class_test c;

    timer.callback = &class_test::method_test;
    timer.myowner = &c;
   TIMER.FINISHED_TIMER();
```

Figure 3. General timer.

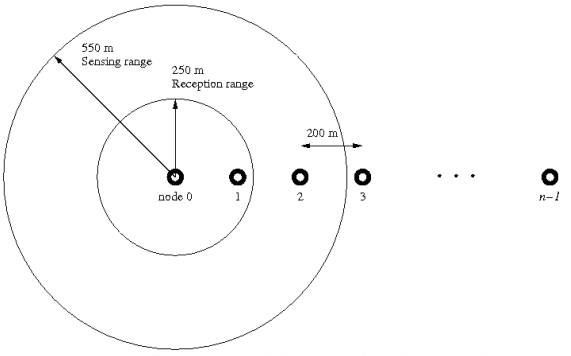

Figure 4. Configuration of the simulated network.

- mac_802_11_rx_coord: the rx_coordination machine. It takes care of the incoming packets. It has the states given below.
  - IDLE: When no packet is being treated by the machine.
  - WAITSIFS_CTS: SIFS period before sending a CTS.
  - WAITSIFS_ACK: SIFS period before sending an ACK.
  - COLL: State assumed when a collision occurs.
- mac_802_11_ch_state: the channel_state machine. It has the states given below.
  - IDLE: No NAV.
  - NAV: NAV timer is running.
- mac_802_11_tx: transmission machine, used to send all the packets. Has an internal Boolean parameter known as tx_active that is TRUE if a packet is being sent or FALSE otherwise.
- mac_802_11: integrates all the state machines and controls the packet receiving (only the transmission), and calls mac_802_11_rx_coord or mac_802_11_tx_coord depending of the nature of the packet (incoming or outgoing). It also takes care of collision detection of incoming packets.

Signals and Event Table. The tx_coordination machine must be warned by the other ones about the events that happen in the MAC. This is done through some standard signals:

- INTERRUPT RECEIVE: when the first bit of a packet arrives from the Physical layer.
- RECEIVED: when the packet that arrived from the Physical layer is fully received.
- INTERRUPT TX: when a CTS or ACK is about to be sent.
- TX DONE: when the CTS or ACK finishes transmission.
- CH IDLE: NAV timer expiration.

Table I. *mac_802_11_tx_coord* state transition.

	INTERRUPT RECEIVE	RECEIVED			INTERRUPT TX	TX DONE	CH IDLE
		no-error		error			
		My packet	Not mine				
IDLE	n	n	n	n	n	n	n
WAIT_ RECVED	*	If will send cts/ack WAIT_TX, else WAIT_IFS_ BACKOFF	WAIT_NAV	WAIT_IFS_ BACKOFF	*	*	n
WAIT_NAV	n	n	n	n	n	n	WAIT_IFS_ BACKOFF
WAIT_IFS_ BACKOFF	WAIT_ RECVED	*	*	*	*	*	*
WAIT_ BACKOFF	If running pause	Resume if won't send cts/ack	n	If NAV off resume	n	resume	resume
WAIT_DIFS	WAIT_ RECVED	*	*	*	*	*	*
WAIT_CTS	n	n	n	n	n	n	*
WAITSIFS_ CTS	*	*	*	*	*	*	n
WAIT_ACK	n	n	n	n	n	n	*
WAIT_TX	*	*	*	*	n	WAIT_IFS_ BACKOFF	n
SENDING_ MPDU	*	*	*	*	*	*	*
BACKOFF_ TXDONE	If running pause	Resume if won't send cts/ack	n	If NAV off resume	n	resume	resume

Legend: * = should not happen, kill execution; n = do nothing

When a signal arrives, the mac_802_11_tx_coord checks its current state, behaving differently for each one. Table I shows the state transition.

New timer mechanism. As explained before, the timers of the original implementation are efficient, but it is hard to understand. In the new module, we use the concept of callback functions.

According to (DEITEL; DEITEL, 2006), it is not possible to create "function pointers" pointing to member methods if the class type is not known. However, if the method is declared as static it is possible to do it. On the other hand, there is another problem: static methods can only access static parameters and methods. The solution is simple: you pass the object as a parameter to the static method, as illustrated in Fig. 2.

At this point, one may see another obstacle: the function pointer uses the class type in the declaration. This way it is not possible to create a general callback pointer. However, to the callback, it does not matter the type in its functionality. Therefore a void pointer as a parameter would work perfectly here.

For the general_timer class, in Fig. 3, the type of object is not important and the solution may work for every situations.

This way of handling timers leads us to an easy understanding code, because when an event needs to be scheduled, there is no need to create a new class. All that have to be done is to set up the method and object called when the timer expires, and obviously, the timer's desired time.

Tests and results

To verify the integrity of the simulator with the new MAC 802.11 module, simulation runs were performed and the results were compared to those obtained using the original implementation. The configuration of the simulated network consists of a sequence (chain) of terminals with two, three, five and ten nodes, each 200 meters away from its neighbours (see Fig. 4). The reception range and the sensing range were set to 250 meters and 550 meters, respectively. Up to 5 CBR (Constant Bit Rate) data flow were set up, always having node 0 as source and node (n-1) as destination, where n is the number of nodes. The flows starting times were randomly selected between 0 and 1 second and each simulation run lasted for 180 seconds.

Table II. Execution time, in seconds. Average of 5 180-second simulation runs.

Number of Nodes	Number of Flows	Average Execution Time [s]		Difference (Ref. Original)
		Original	Proposed	
2	1	0.592	0.594	0.2%
	3	1.781	1.717	-3.6%
	5	2.939	2.865	-2.5%
3	1	1.144	1.106	-3.4%
	3	3.430	3.335	-2.8%
5	1	2.518	2.459	-2.3%
	3	5.128	5.480	6.9%
10	1	6.565	6.217	-5.3%

Table III. Delay, in milliseconds. Average of 5 180-second simulation runs.

Number of Nodes	Number of Flows	Average Delay [ms]		Difference (Ref. Original)
		Original	Proposed	
2	1	2.65	2.35	-11.3%
	3	2.78	2.78	0.3%
	5	4.71	4.36	-7.4%
3	1	5.57	5.59	0.2%
	3	8.42	7.25	-13.9%
5	1	11.4	12.1	-3.6%
	3	597	576	6.9%
10	1	26.2	28.3	-8.3%

Table II presents the average execution time. Although it represents the execution elapsed time, it gives an indication of the amount of processor cycles used in each simulation run. Table III presents the average delay time between source and destination at the application layer; Table IV presents the average packet loss rate; and Table V presents the throughput. It can be seen from these results that the original and the proposed implementations have similar performance.

The results in Table II show that there is no meaningful difference in performance between the two implementations. Tables III, IV and V expose similar results from both simulations, indicating that the proposed module is working properly.

Table IV. Packet loss rate, in percentage. Average of 5 180-second simulation runs.

Number of Nodes	Number of Flows	Average Packet Loss [%]		Difference (Ref. Original)
		Original	Proposed	
2	1	0.0%	0.0%	equal
	3	0.0%	0.0%	equal
	5	0.0%	0.0%	equal
3	1	0.0%	0.0%	equal
	3	0.0%	0.0%	equal
5	1	0.0%	0.0%	equal
	3	45.2%	40.0%	-11.4%
10	1	0.0%	0.0%	equal

Table V. Throughput, in kilobits per second. Average of 5 180-second simulation runs.

Number of Nodes	Number of Flows	Throughput [kbps]		Difference (Ref. Original)
		Original	Proposed	
2	1	49.9	49.9	equal
	3	49.9	49.9	equal
	5	49.9	49.9	equal
3	1	49.8	49.8	equal
	3	49.9	49.8	-0.2%
5	1	49.9	49.8	-0.2%
	3	27.2	29.8	9.6%
10	1	49.8	49.9	0.2%

Conclusion

This article presents a new implementation of the IEEE 802.11 MAC protocol to be used with the network simulation tool NS 2. Its intention is to offer an alternative design that avoids the shortcomings and faults found in the original NS 2 implementation. Results indicate that the proposed code performs as well as the original implementation in regards to the execution time and the protocol performance. However, it adds quality to the simulation tool in the sense that the protocol implementation is closer to the standards, it is easier to understand and modify.

References

AMBLER, S. *The Object Primer*, 2nd. Ed., Cambridge: Cambridge University Press, 2001.

CHEN, Q.; SCHIMIDT-EISENLOHR, F.; TORRENT-MORENO, M.; DELGROSSI, L.; HARTENSTEIN, H. Overhaul of IEEE 802.11 modeling and simulation in ns-2. In: INTERNATIONAL WORKSHOP ON MODELING ANALYSIS AND SIMULATION OF WIRELESS AND MOBILE SYSTEMS, 2007, Chania. *Proceedings*...Chania, 2007. p. 159–168.

CROW, B. P.; WIDJAJA, I.; KIM, L. G.; SAKAI, P. T. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, New York, v. 35, n. 9, p. 116-126, 1997.

CRUZ, E. H. M. *Source code*. Available at: http://perfectsoftware.org/eduardocruz/ns. Access on: 2 Feb. 2008.

DEITEL, P. J.; DEITEL, H. M. *C++ How to Program*. 5th. Ed., Upper Saddle River: Prentice Hall, 2005.

FALL, K. R. Network Emulation in the Vint/NS Simulator. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS, 4th., 1999, Sharm El Sheik. *Proceedings*... Sharm El Sheik, 1999. p. 244-250.

FALL, K.; KANNAN, V. (Ed.) *The ns Manual*. 2006. Available at: http://www.isi.edu/nsnam/ns/doc/. Access on: 2 Feb. 2008.

HENDERSON, T.; ROY, S.; FLOYD, S.; RILEY, G. F. ns-3 project goals. In: 2006 WORKSHOP ON NS-2: THE IP NETWORK SIMULATOR, 2006, Pisa. *Proceedings*... Pisa, 2006.

IEEE STANDARDS. *ANSI/IEEE Standard 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999 Ed., Piscataway: IEEE-SA Standards Board, 2003.

LIU, K. *Understanding the implementation of IEEE MAC 802.11 standard in NS-2*, date unspecified. Available at: http://www.cs.binghamton.edu/~kliu/research/ns2code. Access on: 2 Feb. 2008.

SCHMIDT-EISENLOHR, F.; LETAMENDIA-MURUA, J.; TORRENT-MORENO, M. *Bug Fixes on the IEEE 802.11 DCF module of the Network Simulator ns-2.28*. Tech. Rep: Department of Computer Science, University of Karlsruhe, 2006.