Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 195 (2008) 3–20
www.elsevier.com/locate/entcs

Specification and Verification of the IEEE 802.11 Medium Access Control and an Analysis of its Applicability to Real-Time Systems

Frederico J. R. Barboza [1] Aline M. S. Andrade [2]
Flávio Assis Silva [3] George Lima [4]

Mechatronics Graduate Program
LaSiD - DCC - UFBA
Salvador, Bahia, Brazil

Abstract

The use domain of IEEE 802.11 networks has broadened to several types of application, including those that require quality of service and real-time guarantees. This trend in particular, has motivated the use of formal methods, not only to obtain a more precise knowledge of protocol properties, but also to specify and validate them. In this context, the contribution of this paper is twofold. First, we describe a formal specification of the IEEE 802.11 medium access control functions using UPPAAL, a freeware model checker tool. The described specification allowed us to verify important properties of these functions, taking into account both time and concurrency. Second, we report an experience of model checking a widely used and reasonably complex communication protocol, taking into consideration temporal requirements.

Keywords: Formal Methods, Real-Time Systems, Wireless Network, Software Reliability, Software Specification.

1 Introduction and Related Work

In the past few years protocols based on the IEEE 802.11 specification [8] and applications that make use of it have become increasingly popular. More recently, this wireless network standard has been required to support systems that need quality-of-service (QoS) [16,19] and/or real-time guarantees [17,14]. In this paper we use formal methods to prove some properties of this standard. Since real-time or

[1] Email: fredjrb@ufba.br
[2] Email: aline@ufba.br
[3] Email: fassis@ufba.br
[4] Email: gmlima@ufba.br

doi:10.1016/j.entcs.2007.08.024

QoS oriented applications require high levels of communication reliability, a precise knowledge of the protocol standard properties and its correctness, mainly considering its timing behavior, is needed. Clearly, providing this for a protocol standard with the level of complexity of the IEEE 802.11 is a challenge.

Although the IEEE 802.11 standard is nowadays widely used and model checker tools have become increasingly accessible, the formalization and the formal verification of the standard properties have attracted little attention. A formal description written in SDL (*Specification and Description Language*) can be found in the specification of the standard [10]. This description can be used by simulators to help one to find possible error scenarios [5]. Other approaches based on simulation have already been used [4]. It is well known that simulation is particularly helpful in understanding a complex specification. However, it cannot ensure its correctness.

To the best of our knowledge, some properties of the IEEE 802.11 standard has been proved only recently [18,12]. However, this work has three main limitations. Firstly, although the standard was formally specified, the verification followed an informal and non-mechanized approach. Secondly, as the concept of time was not modelled, only safety and liveness properties (without considering time) were taken into account. Timeliness, as needed by QoS and real-time oriented applications, has not been considered. To understand the third limitation, some explanation about the coordination functions of the protocol standard is needed.

IEEE 802.11 provides two coordination approaches, provided by the Point Coordination Function (PCF) and the Distributed Coordination Function (DCF), respectively [5]. The former uses an arbiter station to coordinate the medium access, making it possible to avoid access conflicts between different stations. The medium access control policy used by the latter, on the other hand, is distributed and it ensures only the best-effort delivery of messages. As systems may alternate between PCF and DCF during its operation time, the protocol behavior must be analyzed considering both these functions in an integrated way. However, [18] have verified these functions in isolation. Issues related to the possible interference that one function may have in the temporal behavior of another has not been taken into account.

Another work presents a formal verification and specification of the medium access control of the IEEE 802.11 using probabilistic model checking. To do this, the authors combine both non-probabilistic and probabilistic models, using timed automata to specify the MAC sub-layer. The probabilistic timed automata is checked by PRISM [11] to verify properties like the probability that a station sends a packet correctly within a certain deadline. However, the coexistence between PCF and DCF was also not considered and sometimes probabilistic answers are not sufficient for applications that require real-time guarantees.

In this paper we present a formal specification and verification of the IEEE 802.11 standard, where the PCF and DCF functions are considered in an integrated

[5] There is a version of the standard which introduces mechanisms to deal with message priorities, called IEEE 802.11e [9]. The main rules of IEEE 802.11e MAC are essentially the same, where the coordination functions, HCF and EDCF have similar meaning as PCF and DCF but operate with traffic categories.

way and timing properties were taken into account. We understand that due to QoS and real-time application requirements, for which communication timeliness may be a reliability issue, applying formal techniques to prove the timing behavior of the protocol is a needed step forward. To do so, we made use of UPPAAL [3], a model checking tool jointly developed by the Basic Research in Computer Science (BRICS) of the Aalborg University and by the Department of Computer Systems of the Uppsala University. UPPAAL provides the concepts of time and clocks, allowing for the modelling, simulation and verification of computing systems that require timing guarantees. UPPAAL has been successfully used to prove correctness of several types of industrial applications [2,7,6], including some communication protocols [15].

Using UPPAAL we have represented the temporal behavior of the IEEE 802.11 standard and modelled the temporal effects of the integration between the PCF and DCF functions. Also, we formally proved important properties of the IEEE 802.11 specification. Among the verified properties we emphasize the support provided by the standard to applications that require timing guarantees. Up to now this kind of property has not been formally proved.

The remainder of this paper is structured as follows. The sections 2 and 3 briefly describe the MAC layer of the IEEE 802.11 standard and UPPAAL, respectively. The specification and verification are presented in section 4. Then, section 5 provides our final comments and points out some possible research directions.

2 IEEE 802.11 Medium Access Control

The IEEE 802.11 standard defines two medium access control schemes or *coordination functions*: *Point Coordination Function* (PCF) and *Distributed Coordination Function* (DCF). The PCF is based on medium access arbitration. One of the stations works as an arbiter, determining when each station of the network is allowed to send frames. The DCF is not based on a special station controlling the access to the medium. Stations trying to send frames contend for obtaining control of the medium. This might lead to colisions.

When PCF is being used to control access to the medium, the network is called to be in the *Contention Free Period* (CFP), since the medium arbitration avoids collisions. When DCF is being used, the network is called to be in the *Contention Period*. Support for PCF is optional. When a network supports PCF, both coordination functions occur alternatively in time.

2.1 DCF - Distributed Coordination Function

The DCF is the basic coordination function in IEEE 802.11 and defines a CSMA/CA (*Carrier Sense Multiple Access / Collision Avoidance*) access method. This method is based on medium sensing before each message is sent (*Carrier Sense*) and on a mechanism for *avoiding* (instead of *detecting*, as in Ethernet) collisions (*Collision Avoidance*). Before sending a frame, a station first verifies if the medium is free (no station is currently transmiting). If it is, the station sends the frame and waits

for an acknowledgment (ACK), confirming correct reception of the message by its destination. After having started, a station continues to transmit a frame until the whole frame is successfully sent, even if collisions might have happened. If the medium is not free, the station executes a *backoff* procedure. After having executed it, the station tries to send the frame again. The station repeats this process until either it receives an ACK frame or it stops after having tried a certain amount of times.

A station verifies if the medium is free or not by using a sensing mechanism in the physical layer together with a logical verification mechanism at the MAC sublayer. In the physical layer, the station senses the physical medium to detect the presence of signals indicating current transmission. In the MAC sublayer, the medium activity verification is performed by using a variable, local to each station, called NAV (*Network Allocation Vector*). At each instant of time, the value of this variable indicates how long the medium will still be busy. The value of this variable is set by using information about the duration of frame transmissions sent by the stations as part of the frame headers. A station only assumes the medium to be free if the medium sensing at the physical layer indicates that there is no signal in the medium *and* if the NAV value is zero.

Before each frame can be sent, the sending station must sense the medium to verify that it is free for a certain period of time, known as *interframe space* (IFS). Three types of IFSs are defined for the DCF: the *Short Interframe Space* (SIFS), the *DCF Interframe Space* (DIFS) and the *Extra Interframe Space* (EIFS). The SIFS is the shortest, followed by the DIFS and EIFS, respectively. The SIFS and the DIFS have both a fixed length for a particular transmission physical medium. The EIFS has a varying length, depending on failure conditions. The IFSs are used as a mechanism to provide prioritary access to the medium. As shorter the IFS, as higher the priority for accessing the medium, since a station needs to wait a shorter time before trying to use the medium. Each IFS is used in specific situations of the protocol.

The backoff procedure works as follows. When a station starts the backoff procedure, it waits for the medium to become free for a DIFS period. After that, it waits for a random period of time, known as the *contention window* or *backoff period*. In order to control how long it still has to wait, each station has a local clock variable, which has its value decremented as the medium is free. If the medium becomes busy during a backoff period, the value of the clock is frozen. The clock is decremented again when the medium becomes free. This procedure is repeated until the whole backoff period has elapsed (the clock reaches zero). Before transmitting, however, the station still needs to sense the medium free for at least a DIFS period.

In order to avoid that collisions occur during the transmission of a large data frame, two channel allocation frames can be used: the *Request to Send* (RTS) and the *Clear to Send* (CTS). To allocate the use of the medium, a station sends a RTS frame to the destination station. When it receives a RTS, the destination station waits for a SIFS and then sends a CTS. After having received the CTS, the source station will wait for a SIFS and will then send its (application) data. The RTS

frame has information in its header about how long the sending station intends to use the medium. When a station reads a RTS frame, it updates its NAV variable with this duration time.

The DCF does not guarantee maximum delay times for frame transmissions (*best-effort* policy) [4].

2.2 PCF - Point Coordination Function

The PCF is defined as optional in the IEEE 802.11 standard and is built on top of DCF. As described previously, the medium access control in the PCF is based on arbitration. The station working as the bus arbiter is called the *Point Coordination* (PC). A station will only transmit if either it was polled by the PC or is replying to a previous transmission with an ACK.

To begin a CFP (period during which the PCF is being used), the PC waits until the medium is free for a time interval (*interframe space*) called *PCF Interframe Space* (PIFS) and sends a special frame, called *beacon frame*. A PIFS is longer than a SIFS and shorter than a DIFS. After having waited for a SIFS period, the PC can choose either to end the CFP, by sending a CF-End frame, or to poll some station, by sending one of the following frame types: CF-Poll, CF-Poll+Data or Data. The first frame is used by the PC to poll a station, without sending additional (application) data. The second is used when the PC polls a station and sends (application) data simultaneously. The third is used only for sending data.

A polled station waits for a SIFS interval and then replies either with a CF-ACK (*acknowledgment*) frame, if it does not have data to send, or with a CF-ACK+Data, otherwise.

The PC starts a new polling cycle by transmiting either a CF-Poll, CF-Poll+Data or CF-ACK+CF-Poll+Data. The CFP ends when the PC sends a CF-End frame.

The frame sent at the beginning of a CFP, the beacon frame, has information about the maximum duration of the CFP (CFP_Max_Duration). Each station, when reading a frame of this type, uses this information to update the value of its NAV. A CFP and a CP might occur during a time interval known as the *CFP repetition interval* (CFP_RATE). Both the CFP_RATE and the CFP_Max_Duration are configuration parameters for the network.

It is to be supposed that a network can support data flows with timing constraints during a CFP [4].

3 UPPAAL

In UPPAAL a system is represented as a state-graph model using a variant of the timed automaton [1]. The properties to be verified are specified in a dense timed logic called *Logic for Safety and Bounded Liveness Properties*($\mathcal{L}_s$) [3], which is defined as a subset of the dense timed logic $\mathcal{L}_v$[13].

3.1 Timed Automata and Timed Automaton Networks

A timed automaton is composed of a set of finite states, an initial state, a finite set of edges and a finite set of clocks. Constants and variables can be declared to represent channels, boolean values or numeric values. Invariant conditions can be associated to the states and three types of labels can be associated to the edges representing: guards, synchronisation functions and actions. The UPPAAL state-graph model is made up of circles that represents states, double concentric circles that represent the initial state, and arrows interconnecting the states representing automaton edges.

A system can be modelled using a timed automaton network in which each automaton models a system part. Clocks are used to specify temporal restrictions which will be associated to the states and edges of the automaton. When the automaton execution starts the clock values increase at a constant rate. Invariant conditions associated with a state restrict the permanence of the automaton on that state.

A *guard* restricts the execution of the edge with which it is associated. A synchronisation function represents either an input or an output operation on a channel. It is defined using the decorated channel variables `channel!` and `channel?` representing the output and input of a message by the channel `channel`, respectively. An edge with a synchronisation function can only be activated synchronously involving two or more automata. An action is represented by a set of values assigned to variables and to clocks and will occur when the edge accompanying it is executed.

When the automaton network execution starts each automaton is at its initial state and its clocks and variables that have not been initialised yet contains zero values. An automaton network configuration is represented by a pair $\langle \overline{q}, v \rangle$, where $\overline{q}$ is a vector which contains the current state of each automaton, and v associates a value with each one of the clocks and variables.

A delay transition corresponds to the passage of time. In this case the automaton network clock values are updated but the automaton current states remain the same. A delay transition can occur only if the clock values do not violate the invariant conditions associated to the states. An action transition occurs by activating one automaton edge in the automaton network. An action transition causes the system to move from configuration $\langle \overline{q}, v \rangle$ to configuration $\langle \overline{q}', v' \rangle$ provided that both the edge guard is satisfied and the synchronisation conditions associated with the edge are executed. Then all the actions in the edge are executed and the clocks unspecified in the actions will not have their current values modified.

Some channels and states can be declared *urgent*. An edge with an urgent channel will be executed as soon as possible, regardless of any delay transitions. Similarly a state declared urgent (marked with the letter `U`) does not permit the evolution of the network by delay transitions.

States can also be declared *committed* (marked with the letter `C`). With such states sequences of atomic transitions can be modelled. In this case delay transitions and interleaving between actions specified in other parts of the network do not occur.

3.2 Timed Logic

The verification of properties in UPPAAL is done by analysing the reachable states of the automaton model starting at its initial configuration. The goal is to verify whether a determined configuration is reachable from the initial states. For this purpose UPPAAL defines a subset of a timed dense logic called $\mathcal{L}_s$. In general this language is composed of logical operators (`and`, `or`, `imply`, `not`), temporal operators (`[]` - always, `<>` - eventually, `-->` - leads to) and path operators (`A` - to all way, `E` - exist a way). In table 1 the syntax and semantic of $\mathcal{L}_s$ are presented informally.

Table 1
Semantic of $\mathcal{L}_s$

Prop.	**Description**
`E<>p`	It is possible to get a state where `p` will hold (possible `p`)
`A<>p`	`p` eventually holds
`E[]p`	Exists a path such that p always holds (potentially `p`)
`A[]p`	`p` always holds (invariable `p`)
`p-->q`	Since that `p` holds `q` eventually holds (`p` leads to `q`)

The symbols `p,q` can assume one of the following values: `Process.state`; `clock` $\sim$ `value`; `p or q`; `p and q`; `not p`; `p imply q`; and `deadlock`.

The variable `Process.state` represents the state `state` of the model `Process`. The variable `value` represents a real positive number. The symbol $\sim$ represents one of the operators in the set $\{=, != , <, \leq, >, \geq\}$ and `deadlock` represents the fact that the system can not progresses without violating some restrictions.

4 Specification and Verification

In this section we describe how the specification and verification of properties for the medium control functions of the IEEE 802.11 were carried out.

4.1 Specification of the IEEE 802.11 Medium Access Control

The MAC sublayer was modeled using six automata. Two of them model the behavior of a station in the DCF. The others model the behavior of a station in the PCF, the behavior of the PC, the behavior of the carrier sense function and the behavior of the medium.

These automata contain a set of global constants and variables: a) to represent parameters that depend on the physical medium (e.g. SIFS, DIFS, PIFS), b) to indicate the transmission time of frames exchanged by the protocol (e.g. ACK_TIME, CF_END_TIME, BEACON_TIME), c) to model the interactions resulting from the frame starting or ending transmission between the workstations and the medium (e.g. channels iniACKm, endACKm), d) to model the interactions resulting from the frame starting or ending transmission between the medium and the workstations

(e.g. channels iniACK, endACK) and e) to control the behavior of the protocol and to indicate its general state (DCF, PCF).

4.1.1 Stations in the DCF

The automata in Figures 1 and 2 model the behavior of stations in the DCF. The former models the task of receiving messages at each station and the latter the task of sending a message.

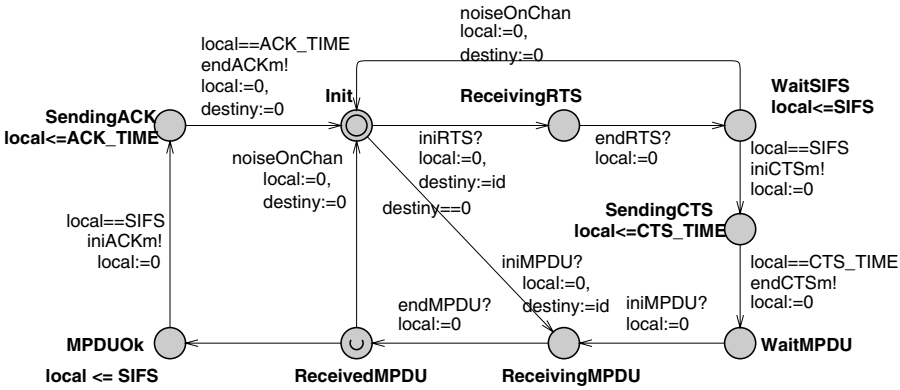

Fig. 1. Message reception at stations in the DCF

To model the receipt of messages a local variable called `local` was declared. This variable is used to control the interframe spacing during a communication process. The behavior of the station specified by the automaton of Figure 1 is as follows: In the `Init` state (the initial state of the automaton) the station is only monitoring the medium, waiting for a message to arrive. A RTS frame is received in `ReceivingRTS`. In state `WaitSIFS` the station waits for the completion of a medium inactivity period to respond to the sending station. A CTS frame is sent in `SendingCTS`. In `WaitMPDU` the station is waiting for the arrival of a message. In `ReceivingMPDU` the station is receiving a data frame. The consistency of a received message is verified in `ReceivedMPDU`. The `MPDUOk` state represents the occurrence of a successful transmission. Finally, in `SendingACK` the station is sending an ACK frame.

The behavior of this automaton can be explained as follows. The automaton starts in the `Init` state and waits for another station to send a RTS frame (edge with `iniRTS?`) or a MPDU frame (`iniMPDU?`). In the first case the automaton begins to process the RTS frame (state `ReceivingRTS`) and at the end of this process it moves to state `WaitSIFS` (edge with `endRTS?`) where it remains for at most a SIFS period, which is guaranteed by the `local<=SIFS` invariant associated with the state.

In the `WaitSIFS` state, the station will randomly choose either to return to the initial state, i.e. RTS was not received successfully, or to answer with a CTS frame. In the latter case, the transition can only occur after a SIFS period has elapsed, which is guaranteed by both the `local==SIFS` guard associated with the edge and the invariant `local<=SIFS` associated with the state `WaitSIFS`. From this

transition, the automaton will synchronize with the medium through `iniCTSm!` starting the transmission of a CTS. The automaton remains in `SendingCTS` until the end of the CTS transmission, signals to the medium that this transmission has finished (`endCTS!`) and waits for the arrival of a MPDU frame. It remains in state `WaitMPDU` until the start of a MPDU transmission (`iniMPDU?`). After this, it makes a transition to `ReceivingMPDU`. In modeling the Basic DCF and DCF with RTS/CTS operations, this state is also reachable from the initial state (state `Init`) through the edge with the same action (`iniMPDU?`).

The automaton remains in `ReceivingMPDU` until it detects the end of the frame transmission (`endMPDU?`) going to state `ReceivedMPDU`. Then, it can randomly choose to either return to state `Init`, i.e. the message was corrupted, or go to state `MPDUOK` if the MPDU reception was successfully. In the latter case, it waits a SIFS period, as described by the invariant (`local<=SIFS`) and by the guard (`local==SIFS`). At this moment, it begins to send an ACK frame through the action `iniACKm!`. Then, the automaton remains in `SendingACK` until the end of the transmission of the ACK frame and finally returns to `Init`(`endACKm!`).

For the task of sending a message, first the backoff procedure (represented by the white circles in Figure 2) is described then the message exchange process. Representing the circles with two colours is only for the sake of clarity.

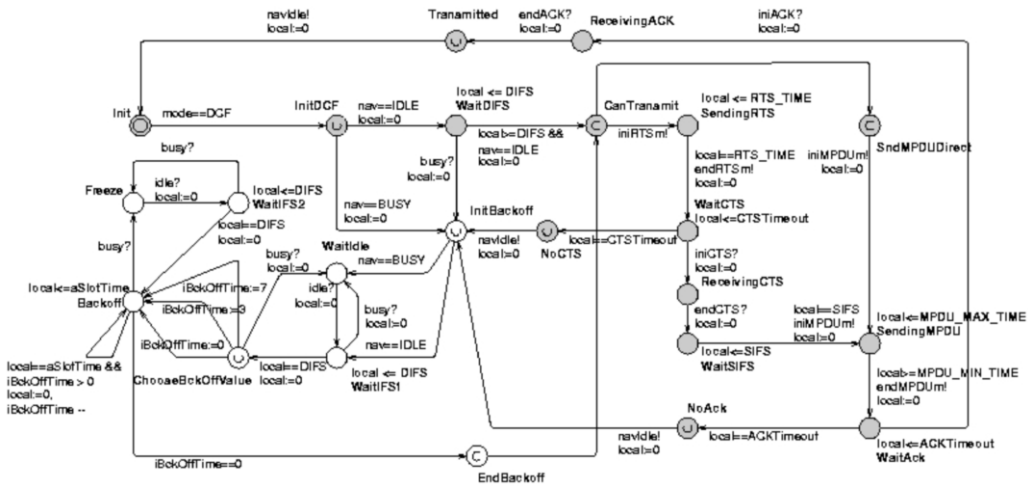

Fig. 2. Message transmission at the DCF

The backoff procedure can be engaged by four edges. The first is from the `InitDCF` state and it is engaged when the station ready to transmit detects the medium busy (`nav==BUSY`). The second edge is from the `WaitDIFS` state and is activated when the station that is waiting for a medium inactivity period elapses (DIFS) perceives that, during this period, another station has begun a transmission (`busy?`). The third edge is activated when the sending station does not obtain a CTS after sending a RTS frame (`NoCTS` state). Finally, the fourth edge (from the state `NoACK`) indicates that a station has not received an acknowledgment frame after transmitting the MPDU because the communication process has failed.

When the station is in backoff, it verifies whether the medium is free or busy. If the medium is busy, the automaton moves to state `WaitIDLE` (edge with guard `nav==BUSY`), where it waits for the medium to be free. Once it is free (`idle?` activated), the automaton goes to state `WaitIFS1`. In this state the station waits a DIFS period of medium inactivity. After this, the station waits the backoff random waiting period. The state `WaitIFS1` can be reached directly from `InitBackoff` if the medium is free when the backoff has been initiated (edge with guard `nav==IDLE`). If the medium is busy during the period that the station is waiting the DIFS period (synchronization `busy?`), the automaton moves to state `WaitIDLE`, where it waits again for the medium to be free. If the medium continues free for the whole period (invariant `local<=DIFS` associated to state `WaitIFS1` and guard `local==DIFS` of the edge that connects `WaitIFS1` and `ChooseBckOffValue` provide it), the state `ChooseBckOffValue` is reached.

The urgent `ChooseBckOffValue` state has several edges to the `Backoff` state. These edges model a random waiting time that is stored in the local clock variable `iBckOffTime` that simulates the backoff clock. These random waiting times were limited for the sake of simplicity, without, however, compromising the model. The automaton remains in `Backoff` at most a $aSlotTime$. If the medium continues free during all the $aSlotTime$, the backoff clock is decreased (action `iBckOffTime--`). However, when waiting for the passage of a $aSlotTime$ in the state `backoff`, a station may perceive that the medium becomes busy again and it should stop its backoff clock. If this is the case, It goes to state `Freeze` where the variable `iBckOffTime` no longer decreases at each $aSlotTime$. The automaton stays in the state `Freeze` until the medium becomes free again and then goes to state `WaitIFS2` remaining there either until the medium becomes busy or until after a DIFS period of inactivity (invariant `local<=DIFS` and guard `local==DIFS`). In the first case, it returns to state `Freeze`. In the second case, it returns to state `Backoff` and waits $aSlotTime$ inactivity period to decrease the backoff clock. When the backoff clock reaches zero (guard `iBckOffTime==0`), it means that the backoff period has finished and the state `EndbackOff` is reached. Then the process of transmission is started immediately from the committed state `CanTransmit`.

The gray circles in Figure 2 refers to message transmission. The station ready to transmit verifies if the medium is free (`nav == IDLE` guard in the edge between the states `InitDCF` and `WaitDIFS`), waits the medium inactivity period (`WaitDIFS` state) and then sends either a RTS frame (`CanTransmit`, `SendingRTS`), modeling a DCF with RTS/CTS or a MPDU frame (`CanTransmit`, `SndMPDUDirect`), modeling a Pure DCF.

The state `SendingRTS` is reached after a write operation in the channel `iniRTSm`. The automaton remains in this state for the necessary time to transmit a RTS frame (invariant `local <= RTS_TIME` and guard `local == RTS_TIME`) and moves to `WaitCTS`. The automaton stays at most `CTSTimeout` time in the state `WaitCTS` and then leaves it because either `CTSTimeout` has elapsed (the case in which the automaton moves to state `NoCTS` and begins a backoff procedure) or a CTS frame has arrived. Exiting from this state occurs through an action with the medium

(`iniCTS?` and then state `ReceivingCTS`). The automaton stays in `ReceivingCTS` until the end of transmission of the CTS frame (`endCTS?`) and moves to state `WaitSIFS`. In this state, it waits the time between SIFS and begins to send the data (`iniMPDUm!`) reaching the state `SendingMPDU`. The period of time that the automaton stays in state `SendingMPDU` depends on the size of the frame that is being transmitted and can vary between `MPDU_MIN_TIME` and `MPDU_MAX_TIME` (invariant `local<=MPDU_MAX_TIME` and guard `local>=MPDU_MIN_TIME`).

In the state `SendingMPDU` occurs the end of the transmission (`endMPDU!`) and then a movement to state `WaitAck`. In `WaitAck`, the automaton waits for an acknowledgment frame (`iniAck?`) or an occurrence of timeout (`ACKTimeout`). If timeout occurs, it is because the MPDU transmission has failed and the automaton has to move to `NoAck` and a backoff procedure should be engaged. If the MPDU transmission has been successful, an ACK frame is received ((`ReceivingACK`)) until it ends (`endACK?`). At this moment, the automaton moves to state `Transmitted` indicating that the transmission has successfully completed, and then moves to state `Init`.

4.1.2 Point Coordinator (PC)

As represented in Figure 3, because of the alternation between CPs and CFPs, the PC needs to wait at least a period given by the repetition rate CFP_Rate minus CFP_Max_Duration to begin a CFP. Thus, only after the passage of a CFP_Rate (invariant `period<=CFP_RATE-CFP_MAX_DURATION` in state `Init` and guard `period==CFP_RATE-CFP_MAX_DURATION`) the PC can try to start the CFP and moves to sate `InitPCF`. In this state the PC checks the medium immediately. If it is free (`nav == idle`)), the automaton moves to state `WaitPIFS` and then waits a PCF interframe space (PIFS).

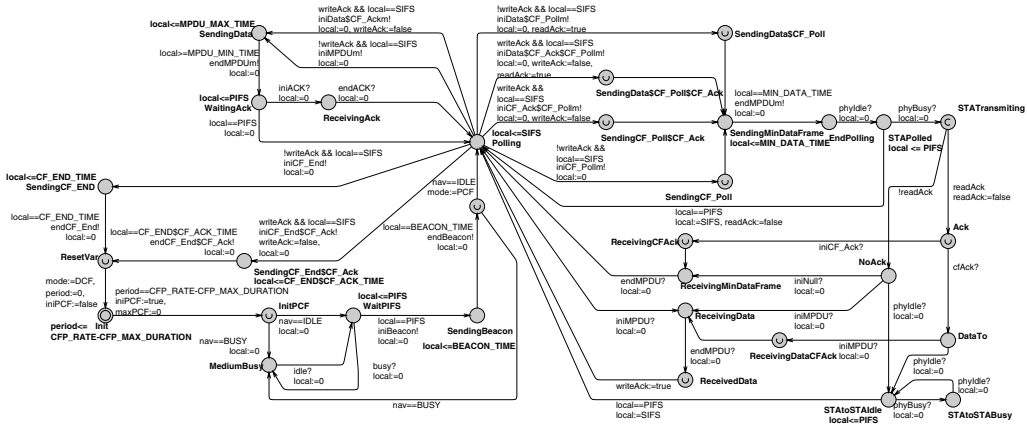

Fig. 3. Point Coordinator

After a PIFS of inactivity, the PC begins the transmission of a beacon frame (`iniBeacon!`) and the automaton moves to `SendingBeacon` and stays there as long as necessary for the transmission of the beacon frame. After this period, the transmission of the beacon is finished (`endBeacon!`) and the PC begins to operate in

PCF mode.

The PC does a series of cycles in which it periodically visits the state `Polling`. In each of these cycle the PC can schedule or not a station and can choose or not to send data to one station. The states `SendingData`, `WaitingAck` and `ReceivingAck` model the sending of data by the PC to a station not scheduled. This behavior is similar to the sending of data in basic DCF mode. The PC can schedule a station in isolation or combined with data and/or acknowledgement frames. These situations are described by the states `SendingCF_Poll`, `SendingData$CF_Poll`, `SendingCF_Poll$CF_Ack` and `SendingData$CF_Poll$CF_Ack`. Depending on the case, the station can answer to the PC with data (`iniMPDU?`), with an acknowledgement frame (`iniCF_Ack?`) or can refuse the schedule (`iniNull?`). In any case, the automaton returns to state `Polling`.

If the scheduled station is ready to transmit to another station, the automaton, depending on the medium state (actions `phyIdle?` or `phyBusy?`) will alternate between `STAtoSTABusy` and `STAtoSTAIdle` states during a medium's PIFS period of inactivity, indicating the conclusion of the communication process between the stations or a timeout.

To finish the CFP, the PC has to wait a SIFS period of medium inactivity in state `Polling` and then sends either a CF_End+CF_Ack frame (state `SendingCF_End$CF_Ack`) or a CF_End frame (state `SendingCF_End`). The automaton then moves to state `Init`.

4.1.3 Stations in the PCF

During the PCF, the station (see Figure 4) leaves the state `Init` for various reasons. The first is when the station processes a MPDU (`iniMPDU?`). In this case the station behaves in a similar way to the stations receiving data in pure DCF, i.e. the station stays at `ReceivingData` until the transmission of the data is complete. The second is when the station is scheduled by the PC. The station moves to state `ReceivingCF_Poll` or `ReceivingDataCF_Poll`, depending on the case, and then to `Pooled`. The scheduled station responds with a) Null (state `SendingNull`), b) CF-Ack (state `SendingCF_Ack`), c) data combined with an acknowledgment (state `SendingData$CF_Ack`) or d) data (state `SendingMPDU`). If the automaton has sent data, it will be acknowledged (state `ReceivingAck`). Then the station returns to state `Init`.

4.1.4 Carrier Sensing Function

The carrier sensing function seen in Figure 5 combines the physical and virtual sensing.

The initial state `Idle` indicates that the medium sensing function has detected that the medium is free. `BusingNAV` and `BusingPHY` are states that represent when the medium is considered to be busy by virtual and physical sensing, respectively. This detection is modeled by read operations (`navBusy?` or `phyBusy?`). After detection, the sensing function notifies the sending station of the current medium situation (`busy!` and `csm:=BUSY`). The detection of the medium as busy by the

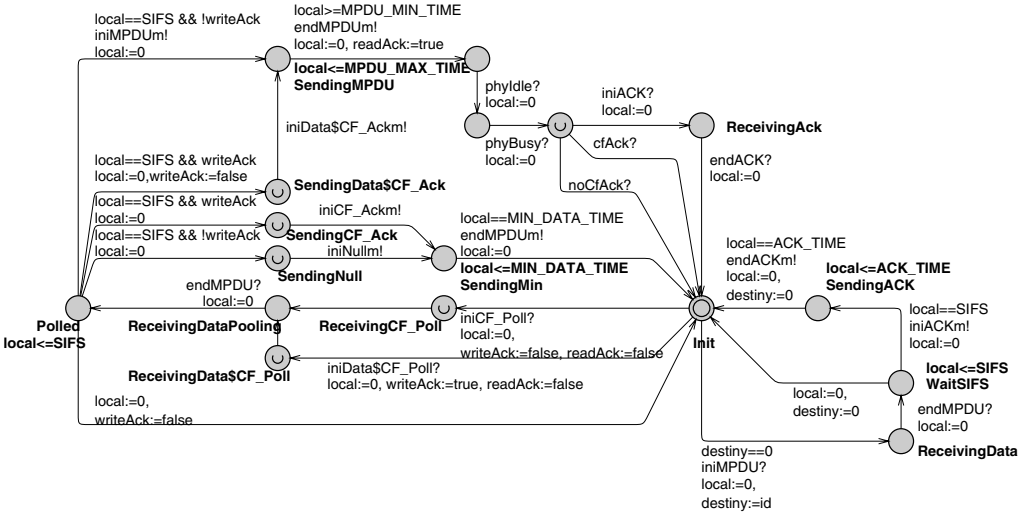

Fig. 4. Station in the PCF

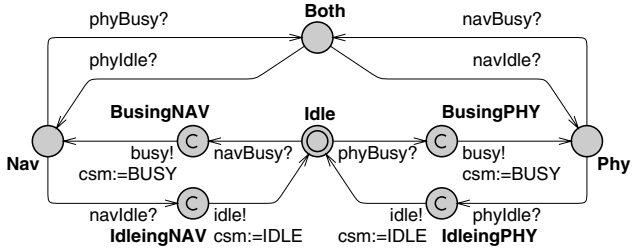

Fig. 5. Carrier Sense Function

virtual and physical sensing is identified by the states `Nav` and `Phy`, respectively. Whichever mechanism is active, the automaton moves to state `Both` if the medium is detected as busy. In returning to state `Idle`, the automaton notifies that the medium is free (`idle!` and `csm:=IDLE`).

4.1.5 Physical Medium

The automaton in Figure 6 models the possibility of collision and breaks down the information that is destinated to distinct stations.

The automaton stays in state `Init` until a station starts or finishes the transmission of a frame. One of the edges that connects `Init` to state `RcvPacket` is activated and the variable `packetMode`) registers which type of event has activated the transition. Although time does not affect state `RcvPacket` (urgent state), interleaving among the automata can occur and collisions are represented by incrementing the variable `collision`. Afterwards, the automaton moves to state `TransPacket` and then to state `NotCollision`, if no collision has been registered (`collision == 0`) or to state `Collision` if a collision has occurred (`collision > 0`).

In state `Collision` the collision counter is decreased to zero and the automaton moves to state `Discard` and then to state `EndFrame` and `Init`.

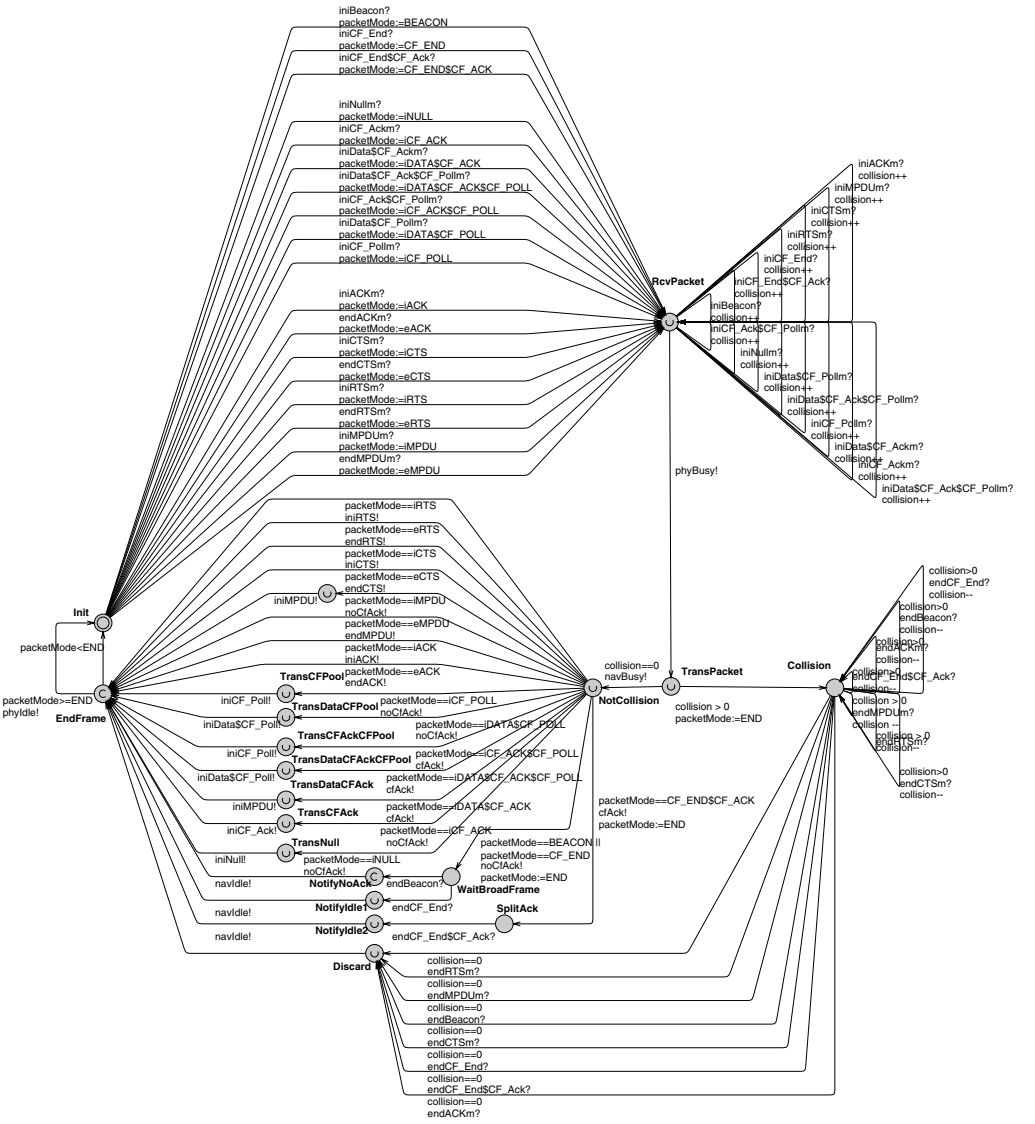

Fig. 6. Physical Medium

If the transmission is of a DFC typical frame, the automaton signals to the receiving station that the frame transmission has started or finished (from state `NotCollision`). This is done according to the variable `packetMode`. Then, the automaton moves to state `EndFrame`.

As regards combined frames the automaton breaks down the acknowledgment information presented in previous frames. To model this behavior, two channels are used: `noAck` and `cfAck`. Thus, if the frame belongs to the acknowledgement family (CF-Ack+CF-Poll, CF-Ack+Data, CF-Poll+CF-Ack+Data, CF-End+CF-Ack), a write operation is done in channel `cfAck`. If the frame is one of CF-Poll, CF-Poll+Data, Null, CF-End, Beacon or MPDU, a write operation in channel `noAck` occurs. The frames that are not addressed to any station in particular (the frames

of broadcast beacon, CF-End e CF-Ack+CF-End) are processed by the automaton.

4.2 Verification

We verified properties of the IEEE 802.11 MAC protocol for several different automata networks. Firstly, the modes DCF and PCF were validated in isolation. Then, we verified properties in scenarios with workstations operating in either mode. As the obtained results for the modes in isolation were compatible with the ones found when they coexist in time, we have only described the latter.

The described results were obtained for a scenario with three workstations operating in DCF mode, two in PCF mode and a Point Coordinator (PC) workstation. This scenario was considered sufficient, since all states are reachable, including those related to the backoff function of the DCF mode. We experienced state explosion when verifying some properties in scenarios with more components.

For the chosen configuration, the automata network was made up of 14 processes. The three workstations running in DCF mode were modelled with the following processes: `dcf1`, `dcf2` and `dcf3`, for modelling the task of sending messages (instances of the automaton shown in Figure 2); `dcfcs1`, `dcfcs2`, `dcfcs3`, for modelling the carrier sensing function (instances of the automaton shown in Figure 5); and `rdcf1`, `rdcf2` and `rdcf3` for modelling the message receiving tasks (instances of the automaton shown in Figure 1). The two workstations running in PCF mode were modelled with the processes `pcf1` and `pcf2` (instances of the automaton shown in Figure 4); the PC was modelled with processes `pc` (instance of the automaton shown in Figure 3) and `pccs`, for modelling the medium sensing function (instance of the automaton in Figure 5); finally, the physical medium was modelled with process `phy` (instance of the automaton shown in Figure 6).

For verifying the properties, UPPAAL 3.4.11 was used running on a Computer Intel Celeron with 2.3 GHz CPU clock and 1GB of RAM. The longest verifying time, approximately 2 hours, was found to verify property 1. The other properties were verified in the range of 15 ms (property 3) to 21 min (property 6).

The properties below specified in $\mathcal{L}_s$ were checked. Please refer to sections 4.1.1 to 4.1.5 for the meaning of the states and variables used.

Property 1 (Deadlock free) *There is no deadlock.*

`A[]!deadlock`. It was checked as true.

Property 2 (Collision occurrence) *Accesses to the medium are subject to collisions.*

This property was verified using the formula `A[]!phy.Collision`. This formula is the negation of the property, i.e., it states that for all paths it is always true that no collision will occur in the physical medium (occurrence of collision is represented by the `Collision` state). It was checked as false.

As stated in property 5, however, collisions do not occur in PCF mode.

Property 3 (Backoff livelock) *If a workstation backs off while attempting to transmit, it may not gain access to the medium again.*

We used the formula `dcf1.InitBackoff-->dcf1.EndBackoff`, which represents the negation of the property. It represents the fact that, for all paths if a process in DCF mode (`dcf1`) initiates backoff (enters the `InitBackoff` state), it eventually completes it (enters the `EndBackoff` state). This formula was checked as false. Consequently, the property is true. If a process does not complete the backoff procedure, it does not obtain access to the medium.

Property 4 (Collision livelock) *A workstation may remain infinitely in collision.*

The formula `dcf1.EndBackoff-->dcf1.Transmitted` represents the negation of the property, i.e., for all paths if a process in DCF mode (`dcf1`) terminates backoff (`EndBackoff` state), then it becomes eventually able to send a message (`Transmitted` state). This formula was checked as false. Hence, there is no guarantee that a station will transmit even if the backoff procedures of previous collisions have finished.

Property 5 (Exclusive medium access in PCF) *In PCF mode, at most one workstation at a time has access control to the medium to transmit its messages.*

The formula `A[](mode==PCF imply !phy.Collision)` asserts it. I.e., while operating in PCF mode, no collision might happen (process `phy` does not enter state `Collision`). This formula was checked as true.

Property 6 (Timely access) *If the PC workstation needs to access the medium, it will do so within a limited time.*

The formula: `pc.InitPCF --> pc.Polling && pc.maxPCF <= (2*PIFS + MAX_DCF_COM_TIME + BEACON_TIME)` was checked as true.

This formula states that (for all paths) if the PC initiates the contention free period (`InitPCF` state), it will eventually reach a state where it polls a station (`Polling` state) and the clock variable `maxPCF` has value equal or less than (2*PIFS + MAX_DCF_COM_TIME + BEACON_TIME). This is the maximum time needed for a station to send a packet of maximum size using RTS/CTS and receive an acknowledgment (see section 4.1.5 for the meanings of these constants). This is an upper limit to the value of `maxPCF` since its value is set to zero when the PC enters the `InitPCF` state. Other workstations (other than PC) do not have guaranteed access to the medium (as stated by properties 3 and 4).

Property 7 (DCF does not interfere in PCF) *If the mode is PCF, a workstation in DCF mode can not use the medium.*

The formula `E<>(mode==PCF && dcf1.CanTransmit)` was checked as false, indicating that in PCF mode a workstation operating in DCF mode (`dcf1`) can not start a transmission (i.e., it cannot enter the `CanTransmit` state).

Property 8 (Reachability property) *Every state of the automaton network is reachable.*

This property was verified using formulas of the form `E<>(`*Process.State*`)`, for each combination of *Process* and *State* of the automaton network. All the variations

were checked as true. It means that all states of all processes of the system are reachable from the initial state of the automaton network. Thus, we considered the presented models as representative of the protocol functionality.

In summary, the above properties describe the main characteristics of the IEEE 802.11 MAC protocol which are: (a) the modes DCF and PCF are deadlock-free (property 1) (b) the DCF mode follows a best-effort policy, since no guarantee about message transmission is given (properties 2, 3 and 4) and (c) even coexisting with DCF, the PCF mode gives timing guarantees (properties 5, 6 and 7). These characteristics indicate the possibility of using the analysed protocol standard for supporting applications with QoS or timing requirements (in PCF mode).

5 Conclusion

We have presented a formal verification and specification of the MAC layer defined in the IEEE 802.11 protocol standard, where both of its coordination functions (DCF and PCF) were considered in an integrated way. These functions have only been considered in isolation up to now.

The considered protocol standard is very complex and required careful modelling work. Using scenarios with a PC workstation, three DCF and two PCF workstations, we have verified several properties of the IEEE 802.11 without facing the state-explosion problem. We believe that such scenarios are representative since all states are reachable. Among the verified properties we emphasize the ability of workstations to have medium access within a bounded known time and so the ability of the IEEE 802.11 to support applications that require timing guarantees. The timing behaviour of the IEEE 802.11 standard has not satisfactorily been considered before in the context of formal verification. Indeed, the results obtained by this work can be used to substantiate the use of IEEE 802.11 protocols in new application domains in which communication timeliness is a reliability issue.

In this version, modeling scenarios where two or more BSAs could overlap total or partially producing collision of beacon frames were not considered. Modeling these situations is a future goal.

Finally, the IEEE 802.11 standard is also often applied to mobile systems, where workstations may move during the system operation. This may generate situations of temporary failures, where workstations cannot be seen by others. Incorporating mobility into the specification should also be part of future work.

References

[1] Alur, R., C. Courcoubetis and D. Dill, *Model-checking for real time systems*, in: *In Proc. 5th Symp. on Logics in Computer Science* (1990), pp. 414–425.

[2] Bengtsson, J., D. Griffioen, K. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi, *Verification of an audio protocol with bus collision using UPPAAL*, in: *Proceedings of CAV'96*, 1996, 1102 of Lectures Notes in Computer Science.

[3] Bengtsson, J., K. G. Larsen, F. Larsson, P. Pettersson and W. Yi, *UPPAAL – A tool suite for symbolic and compositional verification of real time systems*, in: *Proceedings of 1st Workshop on Tools and*

Algorithms for the Construction and Analysis of Systems, 1995, 1019 of Lectures Notes in Computer Science.

[4] Crow, B. P., *IEEE 802.11 Wireless Local Area Networks*, IEEE Communications Magazine (1997).

[5] Doldi, L., "Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis," John Wiley & Sons, Ltd., 2003.

[6] Havelund, K., K. G. Larsen and A. Skou, *Formal verification of a power controller using the real-time model checker UPPAAL*, 1999, pp. 277–298, 1601 of Lectures Notes in Computer Science.

[7] Havelund, K. and K. Lund, *Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL*, in: *Real-Time Systems Symposium, The 18th IEEE*, 1997, pp. 2 – 13.

[8] IEEE, *ANSI/IEEE Std. 802.11* (1999).

[9] IEEE, *IEEE Std 802.11e-2005. Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements to IEEE Std 802.11-1999)* (2005).

[10] ITU-T, *Specification and Description Language (SDL)* (2000).

[11] Kwiatkowska, M., G. Norman and D. Parker, *Probabilistic symbolic model checking with PRISM: A hybrid approach*, in: J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, LNCS **2280** (2002), pp. 52–66.

[12] Kwiatkowska, M., G. Norman and J. Sproston, *Probabilistic model checking of the IEEE 802.11 wireless local area network protocol*, in: H. Hermanns and R. Segala, editors, *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, LNCS **2399** (2002), pp. 169–187.

[13] Laroussinie, F., K. Larsen and C. Weise, *From timed automata to logic and back*, in: *MFCS 95*, 1995, lectures Notes in Computer Science.

[14] Lee, K. C. and S. Lee, *Integrated network of Profibus-DP and IEEE 802.11 wireless LAN with hard real-time requirement*, in: *Proceedings Industrial Eletronics, 2001, Symposium on*, 2001, pp. 227–234.

[15] Lönn, H. and P. Pettersson, *Formal Verification of a TDMA Protocol Startup Mechanism*, in: *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems*, 1997, pp. 235–242.

[16] Sharma, S., K. Gopalan and N. Zhu, *Quality of Service guarantee on 802.11 networks*, , **9**, 2001, pp. 99–103.

[17] Ye, H., G. Walsh and L. G. Bushnell, *Real-time mixed-traffic wireless networks*, IEEE Transactions on Industrial Electronics **48** (2001).

[18] Youssef, M. A., A. Vasan and R. E. Miller, *Specification and analysis of the DCF and PCF protocols in the 802.11 standard using systems of communicating machines*, in: *In Proc. 10th IEEE International Conference on Network Protocols (INCP'02)* (2002), pp. 132–141.

[19] Zhao, L., *A multi-cell dynamic reservation protocol for multimedia over IEEE 802.11 ad hoc WLAN*, in: *Proceedings of the 17th International Conference on Advanced Information Networking and Applications (AINA'03)*, 2003.