Formal Modeling and Verification of an Enhanced Variant of the IEEE 802.11 CSMA/CA Protocol

Youcef Hammal, Jalel Ben-Othman, Lynda Mokdad, and Abdelkrim Abdelli

Abstract: **In this paper, we present a formal method for modeling and checking an enhanced version of the carrier sense multiple access with collision avoidance protocol related to the IEEE 802.11 MAC layer, which has been proposed as the standard protocol for wireless local area networks. We deal mainly with the distributed coordination function (DCF) procedure of this protocol throughout a sequence of transformation steps. First, we use the unified modeling language state machines to thoroughly capture the behavior of wireless stations implementing a DCF, and then translate them into the input language of the UPPAAL model checking tool, which is a network of communicating timed automata. Finally, we proceed by checking of some of the safety and liveness properties, such as deadlock-freedom, using this tool.**

Index Terms: **Carrier sense multiple access with collision avoidance, formal modeling, IEEE 802.11, model checking, unified modeling language state machines, UPPAAL.**

I. INTRODUCTION

The IEEE 802.11 standard specifies the physical and data layers for implementing wireless local area network (WLAN) communication. The data link layer itself is composed of two sublayers:

- A medium access control (MAC) layer that controls the access to the support (through the physical layer) and shares bandwidth between devices;
- A logical link control layer that acts as an interface between the MAC sublayer and the IEEE 802.11 upper layer (network layer).

The IEEE 802.11 standard offers various physical layer implementations, each of which corresponds to a kind of technology that has been commonly used to implement WLAN systems. However, the MAC layer is the same for each implementation, i.e., it defines the exact operation of the carrier sense multiple access with collision avoidance (CSMA/CA) protocol. Since it is impossible to detect collisions during a transmission in wireless communication, we use the CSMA/CA protocol instead of the carrier sense multiple access with collision detection (CSMA/CD)[1] protocol in such a manner that, before a packet transmission, the nodes (i.e., wireless stations) have to listen to the transmission channel (henceforth referred to as a wireless medium) to determine whether other nodes are transmitting. If the medium is sensed as $free$ for a specified amount of time, the node is allowed to begin its transmission. However, if the medium is sensed as $busy$, the node defers its transmission for a random period of time, called a backoff period. The receiving node sends an acknowledge packet (ACK) after waiting for a specified amount of time once the packet is received. If an ACK is not received, the packet is considered lost and a retransmission is arranged.

Note that the IEEE 802.11 MAC protocol defines two forms of medium access: a distributed coordination function (DCF) and a point coordination function (PCF). A DCF is based on the CSMA/CA protocol for sharing access to a wireless medium. A node listens to the medium before a transmission to determine whether someone else is transmitting. Collision detection is not used because a node cannot to hear the medium and transmit simultaneously. In addition to the DCF access method, an optional PCF extension is used, in which nodes in a basic service set (BSS)[2] are polled by the access point, providing an access warranty exists for delay-sensitive applications.

This paper deals with the use of formal methods for modeling and analyzing an enhanced version [1] of the CSMA/CA protocol's DCF mode, in which each station has to disconnect whenever its signal-to-noise ratio (SNR) is lower than a certain threshold. Such disconnections are intended to reduce the number of collisions and improve the transmission rate. We start by building unified modeling language (UML) state machines [2] to thoroughly capture the behavior of wireless stations implementing the DCF, and then translate them into communicating timed automata that are used as the input language of the UPPAAL model-checking tool [3]. Some safety and liveness properties, such as the absence of a deadlock and the successful termination of a transmission, are checked using this tool .

Note that the technique of model checking [4] has become a valuable alternative to simulations and testing, which are intended to explore only some of the possible behaviors and scenarios of a system, leaving open the question of whether unexplored trajectories may contain a fatal bug. Moreover, we chose the use of model checking over other formal methods because it is fully automatic and allows the desired behavioral properties of the enhanced protocol to be checked upon its model through an exhaustive enumeration of all states reachable by the system, as well as the behaviors that traverse through these states. When a design fails to satisfy a desired property, the model checking process always produces a counterexample which demonstrates a behavior that falsifies the property. This faulty trace provides

Manuscript received April 12, 2014.

This work has been supported by Project PHC TASSILI (CMEP) No.12MDU866.

Y. Hammal and A. Abdelli are with the Department of Computer Science, LSI Laboratory, USTHB, Algiers, Algeria, email: {yhammal, abdelli}@lsi-usthb.dz.

J. Ben-Othman is with the Labortoire L2TI, University of Paris 13, France, email: jalel.ben-othman@univ-paris13.fr.

L. Mokdad is with the Laboratoire LACL-CNRS, University of Paris-Est, France, email: lynda.mokdad@u-pec.fr.

Digital object identifier 10.1109/JCN.2014.000068

[1]The CSMA/CD protocol is not used in wireless networks because a mobile node cannot both transmit and receive at the same time.

[2]A BSS is a collection of nodes that have recognized each other and established a communication.

priceless insight into our understanding of the real reason for the failure, as well as important clues for fixing the problem.

To the best of our knowledge, this paper is the first to propose a formal approach covering all steps for the modeling and checking of the CSMA/CA protocol. We use state machines to capture the abstract behavior of the protocol components and translate them into timed automata. The UPPAAL model checker is then fed with the resulting diagrams along with temporal logic formulas specifying the properties we want to check. In fact, despite the advantages of formal methods, only few works similar to our own have undertaken such a formal approach to analyzing the CSMA/CA protocol family; however, even these works have resulted in many different limitations. For instance, the authors in [5] provided a reduced description of the DCF protocol of the IEEE 802.11 standard using finite state machines, and then manually analyzed the models to formally prove that the protocol is free from deadlocks and non-executable transitions. Probabilistic model-checking techniques were used in [6] and [7] for a performance evaluation of various CSMA/CA protocols. The former work analyzes medium access control for sensor networks built on top of the IEEE 802.11 standard. First, Markovian models are built and then analyzed using the PRISM tool, where the properties are specified in probabilistic computation tree logic (CTL). The latter paper models the DCF procedure into probabilistic timed automata. The model is then translated into a finite-state Markov decision process, which is verified using the PRISM tool.

The remainder of the present paper is organized as follows: Section II presents the enhancements brought about by the new variant [1] of the CSMA/CA protocol, and Section III describes the modeling of the CSMA/CA protocol components using UML state machines. Next, in Section IV, we present timed automata that constitute the input language of the UPPAAL model checker and explain how previous state machines are translated into these timed automata. In Section V, we present various properties in terms of the CTL formulae that the UPPAAL checks against the protocol model. Finally, in Section VI, we provide some concluding remarks along with some future directions for our research.

II. ENHANCEMENTS OF THE NEW VARIANT OF THE CSMA/CA PROTOCOL

A quality-of-service (QoS) guarantee for wireless networks is hard to achieve because of the specific characteristics of this type of network. For instance, the radio-link vulnerability attributed to effects such as noise, interference, free-space loss, shadowing, and multipath fading have to be considered. However, MAC protocols developed for these networks do not take these perturbations into account, and most QoS solutions proposed thus far have been limited to the MAC layer and do not exploit information that other layers can provide [8], [9]. Moreover, it was shown that 802.11 suffers from what has been called an "802.11 anomaly" [8]–[10], which has two aspects: Throughputs of all nodes in a 802.11 network fall to the lower level related to the worst node among them, and the bandwidth is divided by the number of mobile nodes connected to the network.

Thus, to overcome this 802.11 anomaly and improve the quality of service of a BSS, cross-layer approaches have been developed [1], [11], [12]. Such approaches are based particularly on information given by the physical layer, some of which comprise basic parameters to ensure a good QoS [13]. In the current paper, we deal with the issue of correctness of the new CSMA/CA protocol variant [1], which also proposed a new cross-layer scheme called adaptive multi-services cross-layer MAC (AMCLM). The goal of this protocol is to improve the QoS of mobile nodes connected in a BSS through a temporary disassociation of those nodes whose SNR is under a defined threshold. In this way, the network's throughput is improved. This new approach aims to improve the QoS of a global network through unselfish decisions of the nodes. In [1] the authors demonstrated the benefit of their method by conducting a performance evaluation of the protocol. For this purpose, they built a discrete Markov chain associated with the behavior of the AMCLM protocol and then analyzed the throughput of the nodes connected to the BSS. The authors computed the throughput saturation and showed that the saturation is much better with the AMCLM protocol than with the CSMA/CA protocol [1].

To check the correctness of the AMCLM protocol, we use a formal method for proving that the new variant of CSMA/CA is deadlock-free and satisfies the safety and other reachability and liveness properties. These properties are rigorously expressed using CTL temporal logic [4], [14], [15]. The first step in our approach consists of translating the informal description of the protocol into UML state machines that accurately model abstract behaviors of the DFC components. These high-level diagrams are then mapped into related timed automata of the UPPAAL model-checker [3], which rigorously check the diagrams in relation to the safety and liveness properties we express through CTL temporal logic formulae. It is worth noting that whenever a property is found to be unsatisfied, the tool provides us with a counterexample, i.e., a model computation path falsifying the property. Analyzing such a counterexample helps us understand the causes of a failure and thereby find solutions for overcoming these causes later.

III. MODELING DISTRIBUTED COORDINATION FUNCTION

A DCF consists of basic access mode as well as an optional request to send/clear to send (RTS/CTS) access mode. Note that, in this paper, we deal with the basic mode of DCF because RTS/CTS mode can be easily seen as a particular use case of DCF owing to the fact that stations exchange special frames using basic mode.

Recall that, before analyzing the DCF, we have to first describe the behavior of the communicating stations (implementing DCF) using UML state machines [2], which are then translated into timed automata of the UPPAAL model checking tool. Using such a gradual approach makes it possible to thoroughly describe the behavior of DCF stations and smoothly generate the input automata of UPPAAL. To this end, DCF basic mode stations are formally seen as a collection of reactive objects (i.e., processes), each of which represents the behavior of a wireless station implementing the DCF basic mode. These processes are depicted using a hierarchical state machine, shown in Fig. 1,

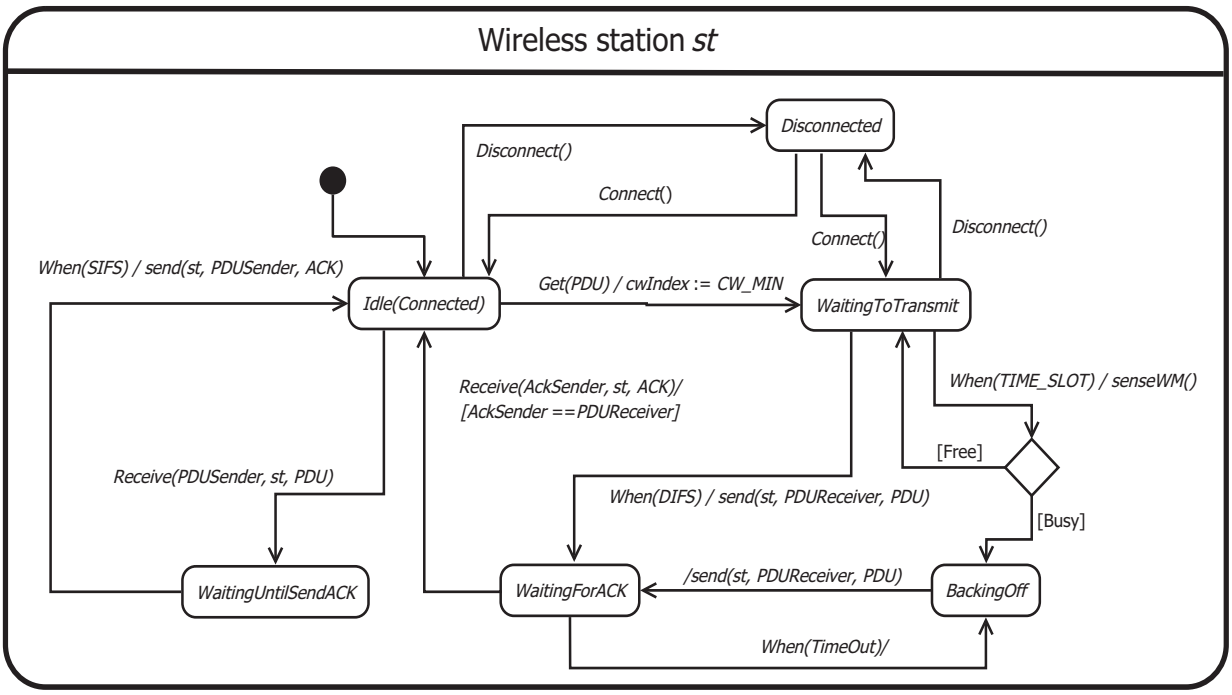

Fig. 1. UML state machine of a wireless station.

which contains a composite state, *BackingOff*, whose nested state machine, depicted in Fig. 2, describes the backoff procedure that the station can invoke according to the CSMA/CA protocol if necessary.

A. A Simplified Variant of UML State Machine

As mentioned in the UML specifications [2], state machines are object variants of Harel statecharts [16]. A state machine shows "a behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions." An event can be a signal, an operation invocation, a time passage, or a condition change, whereas a state is a condition or situation during the lifetime of an object during which it satisfies certain conditions, performs certain activities or waits for a particular event. Here, an event is the occurrence of a stimulus that can trigger a state transition.

Note that a state (also called vertex) may either be simple or composite: Any state enclosed within a composite state is called a substate of that composite state, and when it is not contained within any other state, is called a direct substate; otherwise, it is called a transitively nested substate. When substates can be executed concurrently, they are called orthogonal regions.

A transition (arc[3]) is a relationship between two states indicating that an object in the first (source) state will perform certain actions, and enter the second (target) state when a specified event occurs and the specified conditions are satisfied.

When dealing with composite and concurrent states, the simple term, "current state," can be quite confusing because more than one state may be active at a particular time. If a control is in a simple state, then all composite states that either directly or transitively contain this simple state are also active. Any transition originating from the boundary of a composite state is called a high-level or group transition. If triggered, this results in exiting all the substates of that composite state executing their exit actions, starting with the innermost states in the active state configuration.

Because we only need to use a subset of common features of a UML state machine to describe the CSMA/CA processes, in this paper, we chose to use a simplified and flattened version of such a high-level language where irrelevant complex syntactical constructs are discarded. Hence, we use hierarchical automata such that composite states can only be sequential.

We next provide a formal definition of a state machine as a tuple, i.e., $D = \langle S, Kind, Tag, C, Arcs, s_0, \mathcal{E}, G, \Sigma \rangle$, where the following hold:

- S is a set of states (vertices) with the topmost state, s_0;
- $Kind$: $S \longrightarrow \{SimpleState, CompositeState\}$;
- Tag: $S \longrightarrow \{Initial, Final\}$;
- C: $S \longrightarrow 2^S$ is a mapping that assigns to each state $s \in S$ its direct nested states. $C(s) = \emptyset$ if $Kind(s)$ is a simple state;
- $Arcs \subseteq S \times \mathcal{E} \times G \times \Sigma \times S$: Is the set of transition arcs where $\mathcal{E}$ is the set of events, Σ is the set of (inter) actions, and G is the set of guards. Note that Σ is a set of actions, which may be internal actions or interactions. The latter represent a cooperation between communicating state machines over synchronization channels (i.e., an abstract point of communication). For instance, the tuple (s_1, e, g, a, s_2) denotes a transition arc from state s_1 to state s_2. The transition is triggered by the reception of event e, but this can be taken only if guard g is true, thereby performing action a.

[3]According to the UML terminology, an edge in a state machine is referred to as an arc.

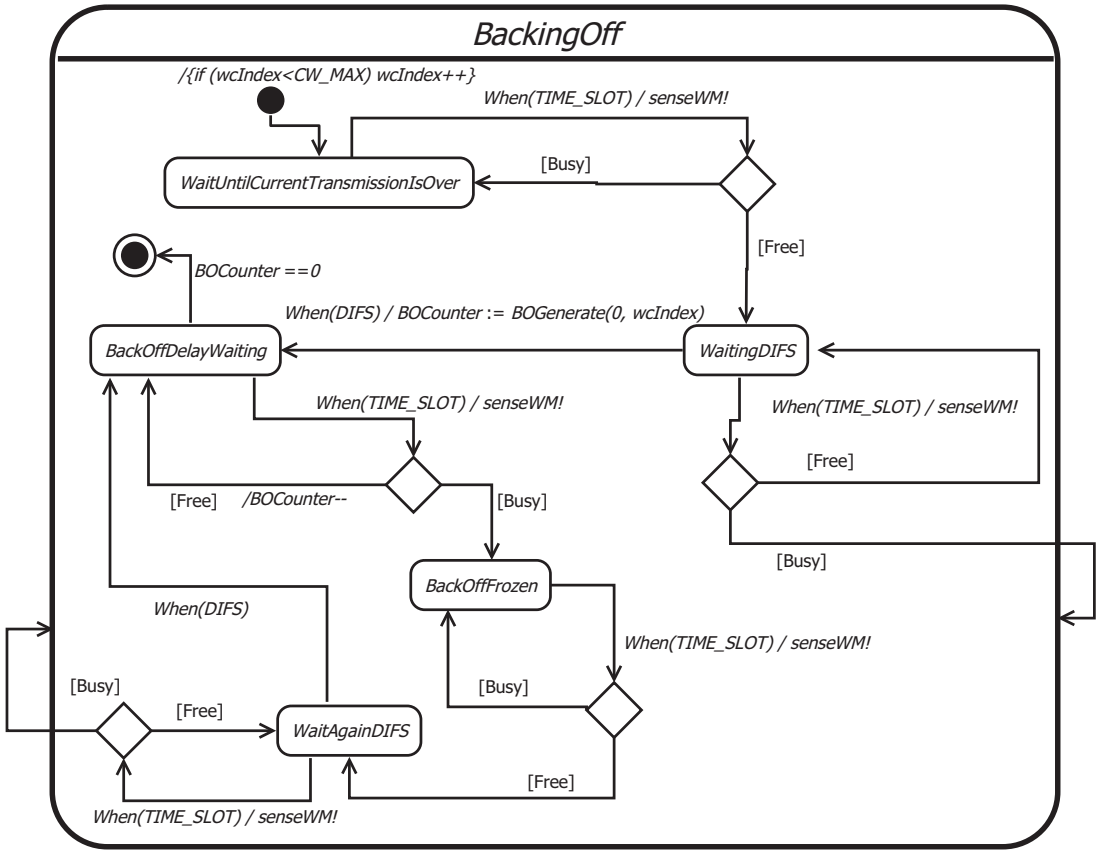

Fig. 2. UML state machine of the BackOff procedure.

B. Distributed Coordination Function of CSMA/CA Protocol

The DCF basic mode states that whenever a wireless station obtains a packet data unit (PDU) to transmit, it has to first sense the wireless medium (WM) to determine whether another station is transmitting before it can initiate a transmission. This is depicted through the state machine shown in Fig. 1, which uses a transition arc outgoing from an *idle* state to a *WaitingToTransmit* state, wherein we check the availability of the wireless medium.

If the medium is sensed as free for a DCF inter-frame space (DIFS) time interval, a transmission will occur. However, if the medium is sensed as busy, the station postpones its transmission until the end of the current transmission, and then invokes the backoff procedure, which lets the station postpone its PDU transmission. As shown in Fig. 1, this part of the DCF process is modeled using a compound transition using a dynamic choice vertex between the *WaitingToTransmit* and *BackingOff* states. The first segment of this transition arc is triggered by a timeout event (i.e., $When(TIME\_SLOT)$), which is generated whenever a time slot is elapsed. The transition is thus enabled and invokes the method for sensing a wireless medium. If the medium is free, the control returns back to the *WaitingToTransmit* state, whereas if the medium is busy, then the control passes to the composite *BackingOff* state, thereby launching its nested state machine, illustrated in Fig. 2. The *WaitingToTransmit* state can also be left when the medium is continually sensed as free until the timeout event *When(DIFS)* is generated, thus triggering the transition to the *WaitingForACK* state. This type of control move causes the PDU to be sent to a receiver station using a method call (leading to an interaction with a wireless medium) named $send(st, PDUreceiver, PDU)$, where st represents a sending station.

According to the DCF, a short inter-frame space (SIFS) is used to give priority access to ACK packets. Immediately upon receiving a packet correctly, the destination station waits for an SIFS interval and then transmits an ACK back to the source station confirming the correct reception. If the source station does not receive an ACK owing to a collision or transmission error, the station reactivates the backoff algorithm after the medium remains free for an extended IFS interval (EIFS). As shown in Fig. 1, there are two outgoing transitions from a *WaitingForACK* state; The transition labeled with $receive(ACKSender, st, ACK)$ depicts the successful termination of the PDU sending interaction once it is triggered by the reception of an ACK from the right expected destination (which is confirmed by the transition guard). The second transition allows the control to return back to a *BackingOff* state, re-invoking the backoff procedure, which is simply triggered by a timeout event (i.e., $When(TimeOut)$), where $TimeOut$ is the sum of the transmission delay and the EIFS.

Likewise, if the station st receives a PDU from another station during an *idle* state, the control moves to a *WaitingUntilSendACK* state, where the receiver station waits until the $When(SIFS)$ event is produced, thus triggering the ACK sending transition back to an *idle* state. As is commonly known,

wireless stations cannot send and receive packets simultaneously.

C. BackOff Procedure

Once the backoff procedure begins, the wireless station will wait during an additional DIFS interval, and then generate a uniformly chosen random backoff delay within the range of $[0, W-1]$, where W is called the backoff window or contention window (CW). The backoff timer is decreased as long as the medium is sensed as idle for a $DIFS$, is frozen when a transmission is detected in the medium, and is resumed once the medium is detected as free again for a $DIFS$ interval. When the backoff counter reaches 0, the station transmits its packet. For IEEE 802.11, the time is slotted within a basic unit of time (i.e., the time slot), which is the time needed to detect the transmission of a packet from any other station. The initial CW is set to $W = CW\_MIN$. If two or more stations decrease their backoff timer to 0 at the same moment, a collision occurs; under this situation, the CW is doubled for each retransmission until it reaches the maximum value, $W = CW\_MAX$, and it remains at that value until it is reset. The value of W is reset after every successful data packet transmission, or when the retry counter reaches its limit CW_RETRY_MAX.

Because W is used to control the backoff counter, its value will affect the performance of the DCF protocol, and improvements can be accomplished by choosing better update rules than those in the IEEE 802.11 standard. For instance, in [1], any station may be disconnected whenever its SNR is lower than a certain threshold. We represent this improvement in the state machine (Fig. 1) using a new state, *Disconnected*, linked to an *idle* state by means of incoming and outgoing arcs labeled with $connect()$ and $disconnect()$ actions, respectively.

Fig. 2 shows how the backoff procedure is performed. The variable $wcIndex$ is used to compute the upper bound of the contention window, $CW = [0, 2^{wcIndex}-1]$. When entering the initial state *WaitUntilCurrentTransmissionIsOver*, the $wcIndex$ variable is incremented (up to the threshold CW_MAX). From this start point, the medium is sensed every time a $When(TIME\_SLOT)$ event is generated. If the medium is sensed as busy (i.e., the current transmission is not yet completed), the control returns back to the initial state; otherwise, it moves into the next state, *WaitingDIFS*. At this point, the state machine tests whether the medium is still free each time a $When(TIME\_SLOT)$ event is dispatched. Thus, either this recursive process continues until the timeout $When(DIFS)$ occurs, causing the control to move into a *BackOffDelayWaiting* state, or the backoff process breaks off when it finds the medium busy. In this case, a high-level outgoing arc to the outer boundary of the *BackingOff* state machine is taken, and thus the backoff procedure is resumed from the beginning and the $wcIndex$ is incremented. At a *BackOffDelayWaiting* state, we measure the time progress using a backoff counter, $BOCounter$, initialized using a random value sampled from the contention window, $[0, 2^{wcIndex} - 1]$. $BOCounter$ is decremented for each time slot if the medium is sensed as free; otherwise, the control enters into a *BackOffFrozen* state, where it stays until the medium becomes free again, thereby passing control to the *WaitAgainDIFS* state. Thereafter, the state machine senses the medium as free again for DIFS, and thus it can return back into a *BackOffDelayWaiting* state and resume decrementing $BOCounter$ counter. In the meantime, if the medium is sensed as busy, the control is passed to the initial state, *WaitUntilCurrentTransmissionIsOver* thanks to the high-level outgoing arc to the boundary of the state machine.

IV. UPPAAL AUTOMATA OF THE PROTOCOL

The UPPAAL model checker [3] is based on the theory of timed automata [17], which are flattened automata augmented with time constraints over logical clocks. However, UPPAAL modeling language offers additional features such as bounded integer variables and urgency. The query language of UPPAAL used to specify the properties to be checked is a subset of CTL. [15], [18].

Accordingly, before a state diagram can be model-checked using UPPAAL, it first has to be translated into its timed automata [3]. We first provide the definition of standard automata, and show how they are augmented using time annotations and a timed semantics to give rise to the modeling language of UPPAAL. Next, through a case study, we describe the method of translating our hierarchical state machines into these timed automata.

A. Timed Automata

Below, we present the definition of a classical finite state automaton, which we use to specify the intended behavior of a process, or an active component (such as a wireless station) without taking time constraints into account:

Definition 1: An untimed automaton is a tuple, $\mathcal{A} = \langle Q, \Sigma, \hookrightarrow, q_0 \rangle$, where the following hold:

- Q is the set of locations (untimed states) of this automaton (depicted as graph nodes in Fig. 3);
- Σ is the set of actions and interactions that this process can perform;
- $\hookrightarrow \in Q \times \Sigma \times Q$ is the set of edges (automaton transitions) between locations (untimed states);
- q_0 is the initial location.

Designers of a reactive component may add any timing constraints to its automaton for interactions that may occur between the component and its environment. Hence, to correctly provide component services to its environment, a sending or reception has to occur in accordance with the timing restrictions. This approach consists formally of the expression of time constraints by means of Boolean formulas over logical clocks. Although such variables express the progression of time, their values can be initialized and tested.

Definition 2: (Timing constraint) Let χ be a finite set of clocks ranging over $\mathcal{R}^{\geq 0}$ (set of non negative real numbers). The set $\Psi(\chi)$ of the timing constraints on χ is defined through the following syntax:

$$\psi ::= true \mid x \ll c \mid x - y \ll c \mid not(\psi) \mid \psi \wedge \psi$$

where $x, y \in \chi$, $c \in \mathcal{R}^{\geq 0}$, and $\ll \in \{<, \leq\}$. Other assertions such as, $x > 3$, $2 \leq x < y + 5$, and $\psi \vee \psi$ can be defined as abbreviations.

The valuation $v \in V$ of the clocks is a function used for assigning a non-negative real value $v(x) \in \mathcal{R}^{\geq 0}$ to each clock,

$x \in \chi$. We can state that v satisfies $\psi \in \Psi$ if $\psi(v)$ is evaluated as $true$. For $v \in V$ and $X \subseteq \chi$, we define $v[X := 0]$ to be the valuation $v' \in V$ such that $v'(x) = 0$ if $x \in X$, and $v'(x) = v(x)$ otherwise. For $\delta \in \mathcal{R}^{\geq 0}$, we define $v + \delta$ to be the valuation $v' \in V$ such that $v'(x) = v(x) + \delta$ for all $x \in \chi$.

A timed automaton is a finite directed graph annotated using conditions and resets over non-negative real valued clocks. We therefore enhance previous untimed graphs using timing constrains by adding three mappings I, G, and Z as follows.

Definition 3: The timed version of automaton $\mathcal{A} = \langle Q, \hookrightarrow, \Sigma, q_0 \rangle$ is an extended graph, $\mathcal{A}_{\mathcal{T}} = \langle \mathcal{A}, \chi, I, G, Z \rangle$, where the following hold:

- χ is a finite set of clocks;
- $I : Q \longrightarrow \Psi(\chi)$;
- $G : \hookrightarrow \longrightarrow \Psi(\chi)$;
- and $Z : \hookrightarrow \longrightarrow 2^{\chi}$.

The first mapping, I, assigns a sojourn or activity condition called an *invariant*, which may be true, to each location of the untimed automaton. The second mapping, G, assigns a timing guard to each edge $(e \in \hookrightarrow)$, which should be true to allow the edge to be taken (i.e., to let the transition fire). The mapping, Z, associates a set of clock initializations, which may be empty (Fig. 3), with each edge.

The state of timed automaton $\mathcal{A}_{\mathcal{T}}$ shows the configuration of the automaton at a particular instant. Formally, this is the pair (q, v) defined based on location q and clock valuation v. In any state, $\mathcal{A}_{\mathcal{T}}$ can evolve either through a change in the discrete state corresponding to movement through an edge, which may change the location and reset some of the clocks, or through a continuous state change owing to the progression of time at the current location. For $a \in \Sigma$ and $\delta \in \mathcal{R}^{\geq 0}$, we define the relations $\xrightarrow{a} \subseteq (Q \times V)^2$ and $\xrightarrow{\delta} \subseteq (Q \times V)^2$ characterizing the discrete and continuous state changes, respectively, as follows:

$$\frac{e = (q, a, q') \in \hookrightarrow, G(e)(v)}{(q, v) \xrightarrow{a} (q', v[Z(e) := 0])}, \quad \frac{\forall \delta' \in \mathcal{R}^{\geq 0}, \delta' \leq \delta . I(q)(v + \delta')}{(q, v) \xrightarrow{\delta} (q', v + \delta)}.$$

The role of the invariants is important. Indeed, as the time progresses, the values of the clocks increase providing that the state satisfies the invariant. For states that do not satisfy the invariant, the time progression is "stopped." This mechanism allows the specification of hard deadlines, i.e., when the deadline specified by the invariant is reached for a certain action, the continuous flow of time is interrupted. Therefore, the action becomes urgent and is "forced" to occur if enabled.

A deadlock status will therefore be given to any timed configuration of an automaton whose related active location has a false activity condition, and whose its outgoing transitions all have false timing guards. In other words, the time cannot progress under such a state in which no actions are enabled in respect to the timing guard. Therefore, there is no method available for leaving this state and enabling the time to progress again.

B. Modeling Language of UPPAAL

The modeling language of UPPAAL consists of networks of timed automata. In fact, a system is modeled as a network of timed automata in parallel, each of which is related to at least one of the system components. As aforementioned, a timed automaton is a flattened finite-state machine extended using clock variables. It uses a dense-time model where a clock variable evaluates to a real number. All clocks progress synchronously, and the model is further extended using bounded discrete variables that are a part of the state. As used in programming languages, these variables are read, written, and subject to common arithmetic operations. A state of the system is defined based on the locations of all automata, the clock constraints, and the values of the discrete variables. Each automaton may fire an edge separately or synchronize with another automaton, leading to a new state.

The UPPAAL modeling language extends the timed automata using the following additional features:

- *Constants* are declared as *const name=value*. By definition, constants cannot be modified and must have an integer value. Bounded integer *variables* are declared as *int*[min,max] *name*, where min and max are the lower and upper bounds, respectively. Guards, invariants, and assignments may contain expressions ranging over bounded integer variables. The bounds are checked upon verification, and violating a bound leads to an invalid state that is discarded (at run-time). If the bounds are omitted, the default range of $-32,768$ to $32,768$ is used.
  A declaration of certain constants denoting the minimum and maximum indices of the contention window, as well as the number of times we can recall a backoff procedure during the transmission of the same packet after the variable $cwIndex$ has reached its maximum value, is as follows:
  `const int` $CW\_MIN = 3$, $CW\_MAX = 10$;
  `const int` $CW\_RETRY\_MAX = 5$;
- *Templates automata* are defined using a set of parameters of any type (e.g., int or chan). These parameters are substituted for a given argument in the process declaration. Herein, we identify stations using the parameter of a new declared type, $idSt$, as follows :
  `typedef int`$[0, NBR\_STATIONS - 1]$ `idSt;`
  where $NBR\_STATIONS$ is an integer constant denoting the number of stations.
- *Binary synchronization channels* (abstract gates) are declared as *chan c*. An edge labeled with $c!$ synchronizes with another labeled as $c?$. A synchronization pair is chosen nondeterministically if several combinations are enabled.
- *Broadcast channels* are declared as *broadcast chan c*. For a broadcast synchronization, one sender $collision!$ can synchronize with an arbitrary number of receivers $collision?$. Any receiver that can synchronize during its current state must do so. If there are no receivers, then the sender can still execute a $collision!$ action, i.e., broadcast sending is never blocking.
- *Urgent synchronization channels* are declared by prefixing the channel declaration using the keyword *urgent*. Delays must not occur if a synchronization transition is enabled in an urgent channel. Edges using urgent channels for synchronization cannot have time constraints, i.e., no clock guards. Urgent locations are semantically equivalent to adding an extra clock x, that is reset on all incoming edges, and having an invariant $x \leq 0$ on the location. Hence, time is not allowed to

progress when the system is at an urgent location.

- *Committed locations* are even more restrictive in their execution than urgent locations. A state is committed if any of the locations in the state is committed. A committed state cannot delay and the next transition has to involve an outgoing edge of at least one of the committed locations.
- *Arrays* are allowed for clocks, channels, constants, and integer variables, and are defined by appending the size to the variable name, e.g.,

  chan $beginSendPDU[NBR\_STATIONS]$;

  chan $endSendPDU[NBR\_STATIONS]$;

  chan $beginReceivePDU[NBR\_STATIONS]$;

  chan $endReceivePDU[NBR\_STATIONS]$;
- *Expressions in UPPAAL* range over clocks and integer variables. Expressions are used with the following labels:
  - ***A guard***: A guard is a particular expression satisfying the following conditions: it is free of side-effects, and it evaluates to a Boolean; in addition, only clocks, integer variables, and constants are referenced (or arrays of these types); clocks and clock differences are only compared to integer expressions; guards over clocks are essentially conjunctions (disjunctions are allowed over integer conditions).
  - ***A synchronization***: A synchronization label is on either the $Expression!$ or $Expression?$ form or is an empty label. The expression must be free of side-effects, evaluate to a channel, and only refer to integers, constants, and channels.
  - ***An assignment***: An assignment label is a comma-separated list of expressions with a side-effect, where expressions must only refer to clocks, integer variables, and constants, and can only assign integer values to clocks.
  - ***An invariant***: An invariant is an expression that satisfies the following conditions: it is free of side-effects, with only a clock, integer variables, and constants referenced, and is a conjunction of conditions of the form $x < e$ or $x <= e$, where x is a clock reference and e evaluates to an integer.

C. Translation of UML State Machines into Timed Automata

The method of translating UML state diagrams into UPPAAL timed automata consists of mapping each state machine into an UPPAAL timed automaton. We also have to formally model the behavior of the wireless medium, be it an access point (in infrastructure mode) or a transmission channel (in ad hoc mode). For both of these situations, any instantaneous synchronized action a between a station and a medium, is represented as two abstract interactions ($a!$ on the sender side, and $a?$ on the receiver side). Moreover, if action a is non-atomic, it needs to be split into two instantaneous sub-actions $begin - a$ and $end - a$, which will be handled as previously described. Therefore, every method call in the state machine of station, st, is translated into a pair of interactions over related synchronization channels. For instance, the sending of a PDU is mapped into the pair ($beginSendPDU[st]!$, $endSendPDU[st]!$), and the message destination is stored in the st^{th} position of the array $To[NBR\_STATIONS]$. Similarly, the reception of the PDU is mapped into the pair ($beginReceivePDU[st]?$, $endReceivePDU[st]?$), and the message source is stored in the st^{th} position of the array, $From[NBR\_STATIONS]$.

We also add a new broadcast channel $collision$ to convey an event broadcasted by the medium to all stations whenever two or more of them try to simultaneously communicate with the medium. We also use an array of channels $fail[NBR\_STATIONS]$, each of which (i.e., $fail[st]$) is related to the signals exchanged between the medium and station st, to depict possible transmission failures.

Before proceeding with the description of the translation method, we provide below the global declarations of the constants and variables used to define our system processes and allowing them to be simulated and verified:

const int $DATA\_RATE = 1375000$; //1.375 MBytes/s.

const int $PREAMBLE = 192$; //length of PDU preamble in bytes.

const int $ACK\_DURATION = PREAMBLE + ((14 * 1000000)/DATA\_RATE)$;

const int $TIME\_SLOT = 20, SIFS = 10, DIFS = SIFS + 2 * TIME\_SLOT, TIMEOUT = SIFS + ACK\_DURATION, EIFS = TIMEOUT + DIFS$;

bool $wmStatus = true$; //a variable that denotes whether the medium is free or not.

int $To[NBR\_STATIONS]$; //$To[st]$ denotes the receiver station to which st is sending a PDU.

int $From[NBR\_STATIONS]$;//$From[st]$ denotes the sender station from which st is receiving a PDU.

int $Duration[NBR\_STATIONS]$;//$Duration[st]$ contains the transmission duration of the PDU sent by station st. Such a value will be computed by a function $getDuration()$ depending on the PDU length.

The following arrays of synchronization channels offer the means to mallow the $Station$, $BackingOff$, and $WirelessMedium$ processes to cooperate through interactions:

chan $fail[NBR\_STATIONS]$;

broadcast chan $collision$;
chan $beginSendPDU[NBR\_STATIONS]$,
$endSendPDU[NBR\_STATIONS]$,
$beginReceivePDU[NBR\_STATIONS]$,
$endReceivePDU[NBR\_STATIONS]$;
chan $beginSendACK[NBR\_STATIONS]$,
$endSendACK[NBR\_STATIONS]$,
$beginReceiveACK[NBR\_STATIONS]$,
$endReceiveACK[NBR\_STATIONS]$;
chan $beginBackOff[NBR\_STATIONS]$,
$endBackOff[NBR\_STATIONS]$;

C.1 *Station Template*

Each state in the UML diagram whown in Fig. 1 is translated into a location using the same label in the target timed automaton (Fig. 3). A transition arc of UML state machine is split into two edges along with an intermediate location if the arc is a compound transition (including a dynamic choice vertex) or is labeled with an action whose performance has a non-zero duration; otherwise, the arc is translated into only a single edge. Moreover, if the compound arc is labeled with an instantaneous action, the in-between location should be a committed location.

The first edge will be labeled with the triggering event of the arc, the second edge with its action, and the intermediate location with its guard. If a trigger event is a timed event (e.g., $When(TIME\_SLOT)$ in Fig. 1), we use clock x to add the invariant $x \leq TIME\_SLOT$ to the source location of the first

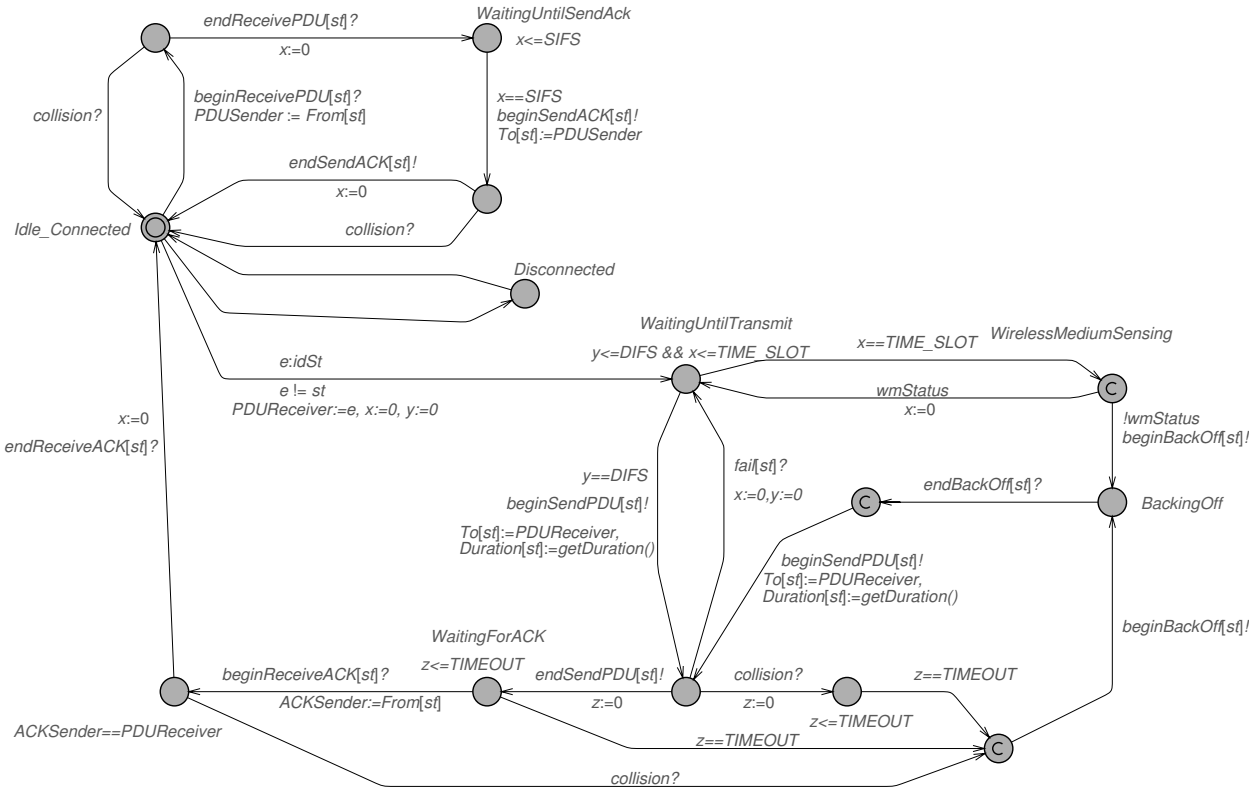

Fig. 3. UPPAAL automaton of a wireless station.

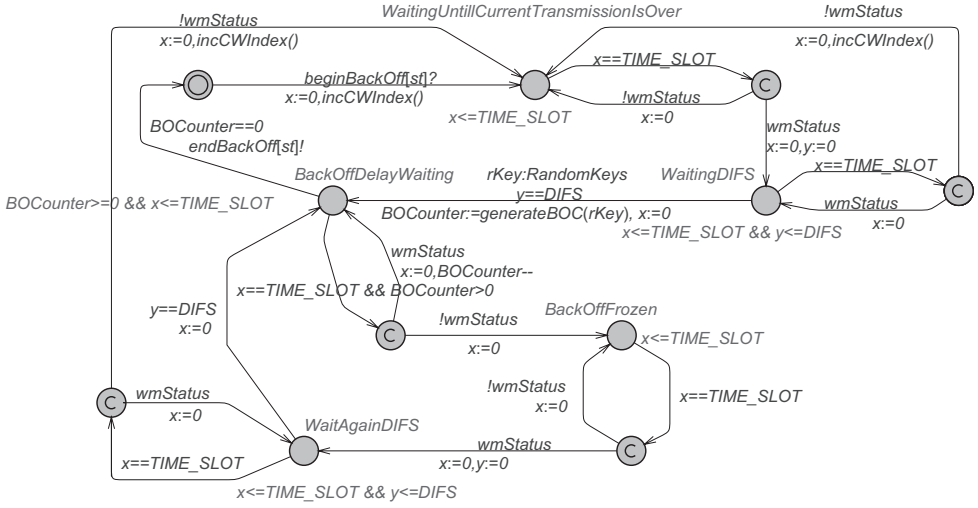

Fig. 4. UPPAAL automaton of the *BackOff* procedure.

edge. We also need to add the initialization of x to all incoming edges for this location (Fig. 3). For instance, the arc between the *idle* and *WaitingUntilTransmit* states in Fig. 1 is translated into a single edge between the corresponding locations in the automaton shown in Fig. 3. The edge is labeled with the assignment $PDUReceiver := e$, where e is the station receiving the PDU. The command e: $idSt$ leads to a non-deterministic choice of the value of e within the interval $idSt$, but thanks to the edge guard it will differ from the sender station, st. The edge also has as its assignments the initialization of clocks x and y used in the invariant $y \leq DIFS \;\&\&\; x \leq TIME\_SLOT$ of the target location *WaitingUntilTransmit*.

On the other hand, each of the two outgoing arcs from the latter state (Fig. 1) splits into two edges (Fig. 3) because one of them is a compound transition and the other requires a non-zero duration. Indeed, the control can remain at location *WaitingUntilTransmit* (Fig. 3) as long as its invariant remains true; however, whenever the guard $x == TIME\_SLOT$ becomes true the Boolean variable $wmStatus$ is tested at the committed intermediate location. If the invariant is found to be false (i.e., the

wireless medium is found to be busy), an initiating signal, *beginBackOff*[*st*], is then sent to the corresponding backoff process, and the control passes to the *BackingOff* location where it awaits the reception of the termination signal, *endBackOff*[*st*]. If the Boolean variable *wmStatus* is true (which means the medium is continuously free), the control returns back to *WaitingUntilTransmit*.

Note that this sensing procedure continues until the guard $(y == DIFS)$ becomes true, thereby causing a control transfer and the sending of the PDU. Because this latter action is not instantaneous, we need to split it into more than one edge (Fig. 3) along with the intermediate (but not urgent) location where either the send action successfully ends, leading the transfer of control to the location *WaitingForAck*, or a *collision?* event occurs, allowing the control to pass to another intermediate location. In either case, a new clock z is initialized and the control waits for as long as the invariant $z \leq TIMEOUT$ remains true. However, in the latter case, the control will be forcibly transferred to the *BackingOff* location, whereas in the former case, the *WaitingUntilTransmit* location can be left to the initial *idle* location if an acknowledgment is correctly received before the timeout event occurs; otherwise, it will be exited to the *BackingOff* location. Recall that this location is entered by sending a *beginBackOff*[*st*] signal to launch the backoff process, and can only be left by receiving an *endBackOff*[*st*] signal.

C.2 *BackingOff Template*

When a *beginBackOff* signal is received at the initial location, a backoff procedure is launched and the control is moved to the *WaitingUntilCurrentTransmissionIsOver* location (Fig. 4), where it remains as long as the wireless medium is sensed as busy. Once the medium becomes free, it should remain so for DIFS units of time so that the control can move forward to the *BackOffDelayWaiting* location; otherwise, the control returns back to the source location, thereby leading to a re-launching of the backoff procedure with a widened contention window.

During the transition to the *BackOffDelayWaiting* location, a backoff counter is randomly sampled from the contention window. The control will stay at that location, and the backoff counter is then decremented for each time slot as long as the medium is sensed as free. Once this counter reaches zero, an *endBackOff* signal is triggered, and the control returns to the initial state. However, if the medium is sensed as busy in the meantime, the backoff counter decrement is frozen until the intervening communication is completed. Thereafter, the medium has to be sensed as free at the *WaitAgainDIFS* location for DIFS units of time to allow the control to return to the *BackOffDelayWaiting* location; otherwise, the backoff is re-launched and the contention window is widened.

C.3 *Wireless Medium Template*

The role of the medium is to forward messages (PDU) from the senders to the receivers, and vice versa, forward acknowledgment packets from the receivers to the senders. Transmissions may fail or interfere with other transmission leading to collisions (Fig. 5). The process of sending a PDU is modeled through two pairs of successive interactions between a medium with a sender and a receiver. The first pair is (*beginSendPDU*[*e*]*?*, *beginReceivePDU*[*e*]*!*) and consists of modeling the beginning of a transmission, whereas the second pair is (*endSendPDU*[*e*]*?*,*endReceivePDU*[*e*]*!*), and models its termination. The delay between these two pairs depends on the duration of the PDU transmission. Such a time constraint is depicted by the invariant $z \leq Duration[from]$ on the node between the two pairs, and the guard $z == Duration[from]$ on the outgoing arc from this node. Any intervening transmission (of a PDU or ACK) during the control remaining in the intermediate nodes will create a collision.

On the other hand, the process of sending acknowledgments is modeled exactly in the same way with only *PDU* suffixes related to interactions' names being substituted with *ACK* suffix.

C.4 *System*

Once the templates have been defined, we build the entire system by instantiating the $Station$ and *BackingOff* templates as many times as necessary, and then combine, in parallel, the produced processes with the single $WirelessMedium$ process (see the following example).

```
Station0 = Station(0);
BackingOff0=BackingOff(0);
Station1 = Station(1);
BackingOff1=BackingOff(1);
...
// List one or more processes to be composed into a system.
system Station0,BackingOff0,Station1,BackingOff1,WirelessMedium;
```

V. MODEL CHECKING PROPERTIES

Model checking is an automatic verification technique for hardware and software systems [4]. Given the model of a system, a model checker automatically tests whether the model meets the given specifications. The specifications are usually written in propositional temporal logic, i.e., linear temporal logic (LTL) or CTL. The verification procedure is an exhaustive search of the state space of the system under design. UPPAAL is one of the many model-checking tools that can be utilized.

Similarly to the modeling process, the requirement specifications must be expressed in a formally well-defined and machine-readable language. Several types of logics exist in the scientific literature, and UPPAAL uses a simplified version of CTL, where the nesting of the path formulae is discarded. As with CTL, the query language consists of both path and state formulae; state formulae describe individual states, whereas path formulae quantify over the paths or traces of the model. Path formulae can be classified into reachability, safety, and liveness types.

- **State Formulae:** A state formula is an expression that can be evaluated for a state without looking at the behavior of the model. For instance, this may be a simple expression, such as $y >= DIFS$, which is a true expression for a state whenever y is greater than or equal to $DIFS$. The syntax of state formula is a superset of a guard formula, i.e., a state formula is an expression free of side-effects; however, in contrast to a guard, the use of disjunctions is not restricted. It is also possible to test whether a particular process is at a given location using an expression in the form of $Station0.Idle\_Connected$, where $Station0$ is the process and $Idle\_Connected$ is the location.

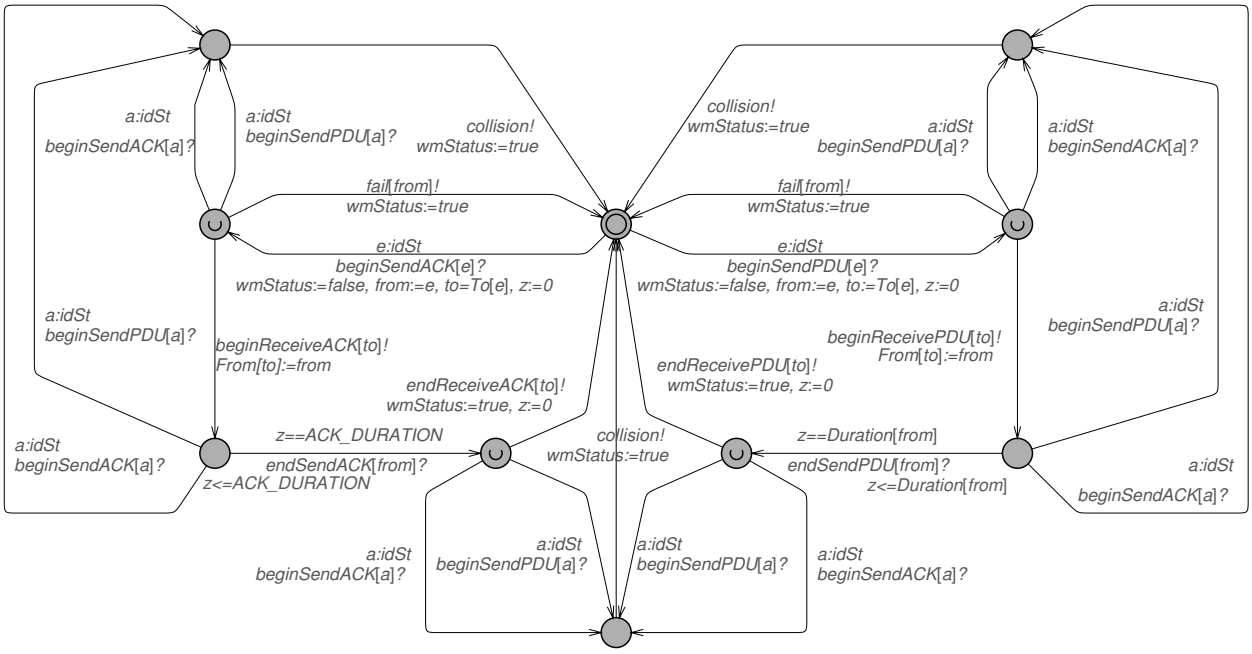

Fig. 5. UPPAAL automaton modeling a wireless medium (ad hoc mode) or access point (Infrastructure mode)

In UPPAAL, a deadlock is expressed using a special state formula (although this cannot be used alone as state formula). Such a formula simply consists of a keyword $deadlock$ and is satisfied for all deadlock states. A state is considered a deadlock state if no outgoing action transitions exist from neither the state itself or any of its delay successors. Owing to the current limitations in UPPAAL, the deadlock state formula can only be used with reachability path formulae, as shown later.

- **Reachability Properties:** Reachability properties are the simplest form of properties. They ask whether a given state formula, ϕ, possibly can be satisfied by any reachable state. Another way of stating this is, does a path starting at the initial state exist such that ϕ is eventually satisfied along that path? We express that some state satisfying ϕ should be reachable using the path formula, $E <> \phi$. Reachability properties are often used to perform sanity checks when designing a model . For instance, for the model described in [1], we checked such properties using formula, e.g., (1),

$$E <> Station0.WaitingForAck \tag{1}$$

to determine whether the enhanced CSMA/CA protocol makes it possible for the sender to send a message at all, and whether it allows a message to be received. Although these properties do not guarantee the accuracy of the protocol by themselves (i.e., whether any message is eventually delivered), they do validate the basic behavior of the model.

- **Safety Properties:** Safety properties are of the form: "something bad will never happen". For instance, a deadlock should never occur in the model of the modified or the original version [1] of the CSMA/CA protocol. A variation of this property is that "something will possibly never happen". For instance when playing a game, a safe state is one in which the game can still be won, and hence will possibly not be lost. For UPPAAL, these properties are formulated positively, e.g., something good is invariantly true. Let ϕ be a state formula. We consider ϕ to be true for all reachable states using the path formula $A[]\phi$, whereas $E[]\phi$ indicates that there a maximal path should exist such that ϕ is always true. For instance, we proved using UPPAAL that the following two properties are satisfied:

$$A[]\ not\ deadlock, \tag{2}$$

$$A[](Station1.WaitingForACK\ imply \\ Station1.y >= DIFS) \tag{3}$$

which state respectively that the model is deadlock-free and behaves in accordance with the $DIFS$ constraint.

- **Liveness Properties:** Liveness properties are of the form: "Something will eventually happen", e.g., when pressing the $on-$button of a TV remote control, the television should eventually turn on; or in a communication protocol model, any message that has been sent should eventually be received. In its simple form, the liveness is expressed using the path formula $A <> \phi$, meaning that ϕ is eventually satisfied. A more useful form is a $leads\_to$ or $response$ property, written as $\phi \dashrightarrow \psi$, which is read as, whenever ϕ is satisfied, then eventually ψ will also also eventually be satisfied, e.g., whenever a message is sent, it will eventually be received[4]. For instance, we proved using UPPAAL that the two following liveness properties are not satisfied:

$$Station0.WaitingUntilTransmit \\ \dashrightarrow Station0.Idle\_Connected, \tag{4}$$

[4] $\phi \dashrightarrow \psi$ is equivalent to $A[](\phi \Rightarrow A <> \psi)$.

$$Station0.WaitingForACK \dashrightarrow Station0.Idle\_Connected. \quad (5)$$

The counterexamples include a peculiar case in which two stations repeatedly try to transmit at the same time and then chose the same backoff delays, thereby leading always to inevitable collisions.

VI. CONCLUSION

In this paper, we presented a formal approach for modeling and checking a new variant [1] of the IEEE 802.11 CSMA/CA protocol. We formally proved that the safety properties are satisfied (e.g., absence of a deadlock) by this new version. However, the model checking shows a peculiar case leading the protocol to not consistently guaranteeing a successful transmission of packets. Furthermore, no upper bound exists for the required delay to successfully transmit a packet.

Probabilistic model checking can therefore be applied to our model to evaluate the timing limits. To enable such an approach, we have to label transitions of our automata with quantitative annotations regarding transmission success, failure, and collision probabilities. Thereafter, more suitable tools must to be used to make use of this information to deal quantitatively with the liveness properties. On the other hand, we believe that our models can also be used for a security analysis, such as attack detection.

REFERENCES

[1] J. Ben-Othman, L. Mokdad, and S. Bouam, "AMCLM: Adaptive multi-services cross-layer MAC protocol for IEEE 802.11 networks," *J. Interconnection Netw.*, vol. 10, no. 4, pp. 283–301, 2009.

[2] Object Management Group, Inc., (2011, Aug.). "Unified modeling language: Superstructure version 2.4.1," [Online]. Available: http://www.omg.org

[3] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Proc. SFM-RT*, pp. 200–236, LNCS 3185, Springer, 2004.

[4] G. J. Holzmann, "Software model checking," *NATO Summer School*, vol. 180, pp. 309–355, IOS Press Computer and System Sciences, Marktoberdorf Germany, Aug. 2000.

[5] A. Vasan and R. E. Miller, "Specification and analysis of the DCF protocol in the 802.11 standard using systems of communicating machines," *Technical Report CS-TR 4358, UMIACS-TR-2002-37*, UMIACS and Department of Computer Science, University of Maryland, May 2002.

[6] P. Ballarini and A. Miller, "Model checking medium access control for sensor networks," in *Proc. ISoLA*, Paphos, Cyprus, Nov. 2006, pp. 255–262.

[7] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol," in *Proc. Joint Intl. Workshop PAPM-PROBMIV*, LNCS 2399, 2002, pp. 169–187.

[8] Y. Chetoui and J. Ben-Othman, "Estimation of the useful channel occupation in 802.11g ad-hoc networks," in *Proc. IEEE GLOBECOM*, 2008, pp. 234–239.

[9] Y. Chetoui, N. Bouabdallah, and J. Ben-Othman, "Improving the bandwidth sharing in IEEE 802.11," in *Proc. IEEE LCN*, 2007, pp. 927–930.

[10] J. Ben-Othman *et al.*, "Facing 802.11 anomaly and improving 802.11 WLANs QoS using a crosslayer design based unselfish behavior," in *Proc. ICN/ICONS/MCL*, 2006, p. 143.

[11] J. Ben-Othman, S. Bouam, and F. Naït-Abdesselam, "802.11 Qos cross-layer rotocol based propagation conditions adaptation," in *Proc. IEEE LCN*, , 2007, pp. 698–702.

[12] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 3, pp. 535–547, Mar. 2000.

[13] J. Ben-Othman, H. Castel-Taleb, and L. Mokdad, "Performance evaluation of mobile networks based on stochastic ordering Markov chains," *Studia Informica Universalis*, vol. 6, No.3, pp. 313–329, 2008.

[14] E. M. Clarke Jr., O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 1999.

[15] T. A. Henzinger, "Symbolic model checking for real-time systems," *Information and Computation*, vol. 111, pp. 193–244, 1994.

[16] D. Harel, "A visual formalism for complex systems," *Science of Computer Programming*, Elsevier, vol. 8, 1987.

[17] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, Elsevier, 1994.

[18] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-checking for realtime systems," in *Proc. LICS*, pp. 414–425, 1990.

Youcef Hammal received the Engineer degree from the Department of Computer Science of the University of Science and Technology Houari Boumediene (USTHB) at Algiers, Algeria in 1992, and the Magister degree from the National Institute of Informatics (INI Oued-Smar), Algiers in 1997. He received a Ph.D. degree in Computer Science from the USTHB University at Algiers, Algeria in 2011. He is Lecturer at Department of Computer Science at the USTHB University where he is also a member of the team Modeling and Verification of Concurrent Systems (MOVES) in the LSI Laboratory. His major research interests include formal modeling and checking of concurrent and reactive systems such as wireless communication protocols, distributed systems, safety-critical, and mission-critical software systems. He proposed formal semantics for UML state machines and interaction diagrams based on timed Petri nets and transition systems so that these could be used as formal specification languages for reactive and concurrent systems. His work deals as well with the issues of these systems correctness and consistency, compatibility, and behavioral substitutability of theirs components.

Jalel Ben-Othman received his B.Sc. and M.Sc. degrees both in Computer Science from the University of Pierre et Marie Curie, (Paris 6) France in 1992, and 1994 respectively. He received his Ph.D. degree from the University of Versailles, France, in 1998. He was an Assistant Professor at the University of Orsay (Paris 11) and University of Pierre et Marie Curie (Paris 6), in 1998 and 1999 respectively. He was an Associate Professor at the University of Versailles from 2000 to 2011. He is currently a Full Professor at the University of Paris 13 since 2011. His research interests are in the area of wireless ad hoc and sensor networks, Broadband Wireless Networks, multi-services bandwidth management in WLAN (IEEE 802.11), WMAN (IEEE 802.16), WWAN (LTE), VANETS, Sensor and Ad Hoc Networks, security in wireless networks in general, and wireless sensor and ad hoc networks in particular. His work appears in highly respected international journals and conferences, including IEEE ICC, GLOBECOM, LCN, MSWIM, VTC, PIMRC, etc. He has supervised and co-supervised several graduate students in these areas. He is widely known for his work on wireless ad hoc and sensor Networks, in particular, security. He is an Editorial Board Member of Elsevier Computer Networks (COMNET), Wiley Wireless Communications and Mobile Computing (WCMC), Wiley Security and Communication Networks (SCN), Inderscience Int. J. of Satellite Communications Policy and Management, IEEE comsoc Journal of Communications and Networks (JCN) and International Journal On Advances in Networks and Services IJANS. He is also an Associate Editor of Wiley International Journal of Communication Systems (IJCS). He has served as a member of Technical Committees of more than 80 international IEEE/ACM conferences and workshops including ICC, GLOBECOM, MSWIM, and LCN. He is a Member of IEEE and ACM. He served as Local Arrangement Chair for the 13th IEEE International Symposium on Computer Communication (ISCC 09). He served as a TPC Co-Chair of IEEE Globecom Wireless Communications Symposium (GLOBECOM 2010) and 9th international Workshop on Wireless local Networks (WLN09) and 10th international Workshop on Wireless local Networks (WLN10). He served as a Publicity Chair of several conferences such as the 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 09), IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WOWMOM 2010), and 25th Biennial Symposium on Communications. He has served as TPC Co-Chair for IEEE GLOBECOM Ad hoc and Sensor and and Mesh Networking (Globecom 2011, 2014), 6th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet 2010, 2011, 2012), Wireless Networking Symposium of the 7th International

Wireless Communications and Mobile Computing Conference (IWCMC 2011, 2012, 2013, 2014), IEEE International Conference on Communications Ad hoc and Sensor and and Mesh Networking (ICC 2012, ICC 2014). He has served for other conferences in ICNC, WSCP, and CNIT. He has also served as Tutorial Chair for Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2014). He was the secretary and he is currently Vice Chair of the IEEE Ad Hoc and sensor networks technical committee since Jan. 2012. He is an Active Member of IEEE CIS-TC and WTC.

Lynda Mokdad received her Ph.D. in Computer Science from the University of Versailles, France in 1997. She was Associate Professor at University Paris-Dauphine from 1998 to 2009. She is currently Full Professor at University Paris-Est, Créteil since 2009. Her main research interests are about performance evaluation techniques and applications in wired, mobile and wireless networks, Ad hoc networks, and in software technologies as Web services. She is recipient of the Best Paper Awards of IEEE International Conference on Communications and Information Technology (ICCIT 2011) and IEEE International Conference on Communications (ICC 2011). She has served as Technical Committee for more than 50 international IEEE/ACM conferences and workshops including ICC, GLOBECOM, MSWIM, LCN, etc. She is a Member of IEEE and ACM. She also served as a TPC Co-Chair of Communication Software symposium at IEEE International Conference on Communications (ICC'13), Ad hoc and Sensor Networks symposium at GLOBECOM 12; 5th and 6th IEEE International Workshop on Performance Management of Wireless and Mobile Networks (P2MNET 09 and 10); She is the Founder and Chair of IEEE Performance Evaluation of Communications in Distributed Systems and Web based Service Architectures (PEDISWESA) which arrives this year at the sixth edition. She was Co-Chair of 11th international Workshop on Wireless local Networks (WLN 11) as well. She served as a Publicity Chair of 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 10) and as Awards Chair of the 8th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-10), She serves as Editor of Wiley International Journal of Communication Systems (IJCS) and Wireless Communications and Mobile Computing (WCMC). She was Secretary of IEEE Communication software (CommSoft) and she is currently serving as a Vice Chair. She is Active Member of IEEE Ad hoc and sensor Networks.

Abdelkrim Abdelli is an Associate Professor at the LSI laboratory of the University of Science and Technology Houari Boumediene (USTHB) at Algiers, Algeria. He received from the same university both his Ph.D. and its HDR degrees in 2007 and 2009, respectively. His research topics deal mainly with multimedia systems and formal verification of real time systems.