Specification and Analysis of the DCF Protocol in the 802.11 Standard using Systems of Communicating Machines

Arunchandar Vasan, Raymond E. Miller
UMIACS and Department of Computer Science
University of Maryland
College Park, MD 20742.
Email: {arun,miller}@cs.umd.edu

CS-TR 4358, UMIACS-TR 2002-37
May 2002

Abstract—**The 802.11 specification is an emerging standard for WLANs. In this paper, we propose a formal model for a section of the 802.11 MAC protocol using systems of communicating machines. We model the ad-hoc mode of the DCF, i.e., CSMA/CA protocol and the MACA[1] using RTS/CTS sequences. Each station is modelled as a finite state machine which has a set of local variables, and the Wireless Medium is modelled as a shared variable. Analyses show that the 802.11 MAC CSMA/CA protocol and the MACA using RTS/CTS exchanges are free from state deadlocks and non-executable transitions. However, the MACA protocol has a potential livelock, though it is unlikely it will come to pass in normal operation.**

Keywords—**802.11, Communicating Machines, Formal Models, MACA, CSMA/CA**

I. Introduction

802.11 [1] is an emerging IEEE standard for Wireless LANs (WLANs) and is primarily meant for use in a limited geographical area. The main issue addressed is mediating access to a shared communication medium- in this case, the Wireless Medium (WM) of electromagentic waves travelling through space.

In this paper, the basic Distributed Coordination Function (DCF) operation of the 802.11 MAC protocol (CSMA/CA) and the Medium Access with Collision Avoidance (MACA) protocol using RTS/CTS exchanges are formally specified using a system of communicating machines (SCM) and analyzed. By ad-hoc, we mean that the stations communicate with each other directly rather that using an Access Point (AP), which is otherwise possible. We model the DCF in ad-hoc mode as the access point behaves like an ordinary machine, i.e., it does not do anything central in DCF.

We demonstrate that the CSMA/CA and the MACA protocols are free from deadlocks and non-exectuable transitions. However, the MACA protocol is shown to have a potential livelock, which may not be seen in practice.

Each station in the wireless network is modeled as a machine which consists of a set of states and local variables. Communication between these machines is achieved by shared variables. A machine moves between its states using transitions. Each transition has an enabling predicate and an action. The actions modify variables shared globally across machines as well as variables local to a machine. The shared variables provide a powerful setting for capturing the broadcast network, i.e., the wireless medium shared by all the stations.

The contributions of this paper are casting of the DCF and the MACA protocols of the 802.11 MAC protocol in a formal setting and the analyses for safety. Ideas used in the extension of SCMs with timing, introduced in [3], are used for capturing timing specifications like timeouts.

The primary reasons for choosing SCMs for modelling are the flexibility provided by variables with actions à lá programming languages and the formalism offered by finite state machines. Simultaneous transitions are permitted in SCMs unlike pure finite state machines, and this is used to model simultaneous access to the WM by different stations. Data transfer is modelled using shared variables, which would be very difficult with FIFO queue like models owing to the broadcast nature of the medium.

In section 2, we briefly review some terminology from the 802.11 standard. We follow this with a review of Systems of Communicating Machines model in section 3. Section 4 introduces the variables used in our model and outlines our assumptions in abstracting reality. Sections 5 and 6 contain the specification and analyses of basic DCF and MACA respectively. We discuss our results in section 7 and conclude in section 8.

II. The 802.11 MAC layer protocols

We review some of the terminology used by the 802.11 specification in this section. A station refers to a host using the Wireless Medium (WM) to communicate. A basic service set (BSS) is a set of stations that communicate with each other using an access point (AP), which is possibly connected to a wired LAN. An Extended Service Set (ESS) is a set of BSSs and, possibly, wired LANs connected as a single service set through a Distribution System(DS). An Independent Basic Service Set (IBSS) is a set of stations that communicate directly without making use of an AP.

A coordination function is a logical function that determines *when* a station can send Protocol Data Unit (PDU) frames, i.e.,

[1] The 802.11 standard does not refer to RTS/CTS exchanges as MACA. However, the paper which introduced this idea called it the MACA, and we use this name.

how the medium is shared. This can either be a distributed coordinated function (DCF) or a point coordination function (PCF). The DCF achieves coordination by running a distributed algorithm i.e., all stations run the logic of the algorithm. The PCF achieves coordination by a centralized algorithm i.e., only a single station within the BSS runs the logic of the algorithm. The AP runs the logic for Point Coordination.

A. DCF Operation

The basic DCF access method for the 802.11 MAC is Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). The carrier sense can be done through physical and virtual mechanisms. A station senses the medium to check if another station is transmitting. This is the physical mechanism. Alternately, the virtual carrier-sense is achieved by distributing reservation information along with RTS/CTS Exchanges in MACA (described later).

If the WM is not busy, the station may proceed to transmit. Before transmission it waits for a DIFS duration. If the medium were to remain free for a DIFS, the station proceeds to transmit the DATA frame.

If the medium becomes busy during the DIFS time interval, the backoff procedure is enabled. Similarly, if the medium were found busy when sensing for the first time, a station enables the backoff procedure.

When the backoff procedure is invoked, the station waits till the current transmission is over. After the end of the current transmission, the station waits for an amount of time equal to the DIFS. Once the medium was detected to be idle for a DIFS, the station performs the additional backoff wait before actual transmission. It sets a timer to some randomly chosen value in a specified interval. If the medium is free throughout this interval, and the timer expires, the frame is transmitted. If the medium becomes busy during this interval, the timer is *frozen* at its current value but *not reset*, and the station waits for the medium to become free, waits for an DIFS, and performs the backoff wait again. This process continues till the backoff timer finally expires, and the station transmits the frame.

When a station receives a DATA frame, it waits for a SIFS (Short IFS) duration, and transmits an ACK to the sender. There is no carrier sensing or backing off for ACK frames. The sender schedules a retransmission if no ACK is received within a specified timeout duration.

B. DCF with RTS/CTS Exchanges : MACA

In this mode, the sender and receiver exchange special frames called Request To Send (RTS) and Clear To Send (CTS) before transmission. Other stations observe this exchange, and do not transmit in the meanwhile, thereby reducing the number of collisions. This protocol is called the Medium Access with Collision Avoidance (MACA). It also solves the *hidden node* and the *exposed node* problems specific to wireless networks [1],[2]. The RTS/CTS frames have duration fields which indicate the time for which the medium would be busy to other nodes in the neighborhood of both the sender and receiver. If a machine receives an RTS/CTS of duration time τ at time t, it marks the medium as busy in its NAV (Network Allocation Vector) for the time interval $[t, t+\tau]$. In addition, any directed data frame, i.e., a frame which has a unicast destination address has a duration field which can be used to announce the time that the medium is reserved, to the end of the immediately following ACK.

The MACA protocol provides for resource reservation for a short duration of time. A station which needs to send data sends an RTS (Request to Send) frame in the normal CSMA/CA style. The RTS frame has two octets, which specify the time for which the medium is reserved for the subsequent data frame and its ACK to be transmitted. All stations which can hear the sender observe this, and do not make any attempt to transmit for this time interval.

The receiver when it receives the RTS frame, sends a CTS frame *after waiting for an SIFS* which reserves the medium using the same two octets. All stations within hearing distance of the receiver observe this, and they do not make any attempt to transmit for this timer interval.

The sender now sends its data frame after the elapse of an SIFS, without backing off. This is because all the nodes in the neighborhood would have heard the RTS/CTS exchange. Likewise, the receiver, on receiving the data frame responds immediately with an ACK after waiting for a SIFS.

We note that once the RTS has been transmitted successfully, the receiver *does not* perform backoff to transmit the CTS or the ACK frame. Once the sender receives the CTS, it *does not* perform the backoff for transmitting the data frame either.

III. Systems of Communicating Machines

We review the basic formalism used in systems of communicating machines. The interested reader is referred to [6] for more details.

All sets are finite in the following description. A system of communicating machines is an ordered pair $C = (M, V)$ where M is a set of machines, and V is a set of shared variables. For each machine i, there exist two subsets of V namely R_i and W_i which contain the variables which machine i has read access and write access respectively.

Each machine $m \in M$ is a 5-tuple given by (S, s, L, N, τ) where:

- S is the set of states in the machine.
- s is a specially designated state called the starting state.
- L is the set of variables local to the machine.
- N is a set of names, each of which is associated with a unique pair (p, a) where p is a predicate on the variables of $L \cup R_m$, and a is an action on the variables of $L \cup R_m \cup W_m$. An action is a partial function from the values of the local and read action variables and write action variables.
- $\tau : S \times N \rightarrow S$ is a transition function which is a partial function from the states and names to the set of states.

The system state tuple is the tuple of all the machine states. The global state is the system state tuple taken along with the values of all the variables. The system state is the system state tuple taken along with all the enabled transitions. A transition is enabled when the predicate associated with it evaluates to be true.

All machines start in their respective start states. Intuitively, a transition is *enabled* when the predicate it is defined on becomes true, and the associated action changes the variables it can modify, i.e., those in W_i, as an effect. These changed variables will now enable other transitions, and the machines proceed to move between the states.

IV. External Variables[2] and Assumptions in the model

We have a boolean variable called *HaveData* which is set to true by a higher layer, when there is data to be sent. In addition, the destination address of the current frame is available in *Address*. PCF_PERIOD (duration of the PCF period), SIFS, PIFS, ONE_POLL_T (time needed to send a poll and elicit a response), BEACON_PERIOD(time between beacons) are other variables which are set by entities outside of our specification. We model the medium as a shared variable with the following fields.

- Duration: for the DCF protocol, stores the remaining time in the current reservation. In the PCF protocol, it stores the remaining time in the PCF period.
- SA: A field to store the source address of the message on the medium.
- DA: A field to store the destination address of the message on the medium.
- Beacon: set to 1 if the message on the medium is a *Beacon* message.
- RTS: set to 1 if the message on the medium is a RTS messages.
- CTS: set to 1 if the message on the medium is a CTS messages.
- ToDS: set to 1 if the data message is from a station, set to zero otherwise.
- End: set to 1 if the message on the medium is an *CF_END* message, i.e. end of contention free period.
- Poll: set to 1 if the access point wants to poll a station.
- NullData: set to 1 if the message on the medium contains no data.
- Ack: set to 1 if the message on the medium contains an acknowledgment for a previously sent data message.

Some messages are broadcast messages, e.g. the *Beacon* messages. In our specification, these messages have a destination address set to *BROADCAST*. In order to model this situation, we make the access point ensure that all entities hear the broadcast before it clears the medium. This is accomplished by using a shared array *AllHeard[0..NSTA- 1]* that has an entry for each station. The entry for station *i* is set to *true* by the station if this station heard the last broadcast by the access point. When all the entries of the *AllHeard* array are *true*, the access point clears the medium and the operation of the protocol continues. Figure 2 showsthe *Medium and AllHeard* shared variables.

Assumptions

We make the following assumptions in our model:

- We model unicast traffic from station to station.
- We model the DCF in the IBSS mode and the PCF in the Infrastructure mode with an AP.
- In DCF mode, the protocol handles errors by retransmissions. In the PCF mode, we assume that there are no transmission errors.

[2]We do not use some of the fields of the shared Medium and some of the variables described here in our report. They have been included to maintain commonality with [7], and are used in [7]

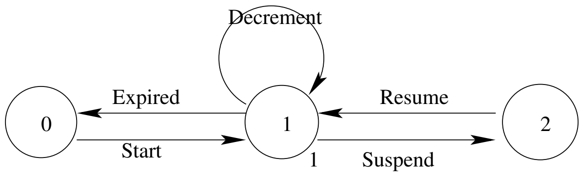

Fig. 1. State description of a generalized timer

Transition	Predicate	Action
Start	st=ACT	
Expired	val=0	st:=EXP
Suspend	st=INACT	
Resume	st=ACT	
Decrement	st=ACT $\wedge$ val > 0	val:=val - 1

TABLE I
TRANSITIONS FOR A GENERALIZED TIMER MACHINE

- In DCF with RTS/CTS, i.e., MACA, all machines are assumed to implement MACA or all machines implement just basic DCF.
- We do not model Extended Service Sets with more than one AP.
- We do not model buffering of data at the AP.
- The only type of management frames we model are the *Beacon* and the *End* frames which are broadcast by the AP to signal start and end of the Contention Free Period.

V. Formal Specification of the CSMA/CA Protocol

We generalize the timer machine introduced in [**?**] to accomodate two more transitions called *Suspend* and *Resume*. *Suspend* stops the machine from counting down, and *Resume* allows a suspended machine to resume counting. Each machine modeling a station has a number of timer machines. Each station-machine can alter the state of its timer machines by starting, suspending, and resuming it.

The timer machine starts out in state 0. Each timer machine has two variables: *st* and *val*. The *st* variable, standing for status, can be one of ACT, INACT, and EXP. ACT stands for the timer being active, i.e., ticking. INACT stands for the timer being suspended, while EXP stands for the expiry of the timer. The *val* variable stands for the remaining time before the timer expires. These two variables are shared between a timer machine and the station machine which owns the timer machine. A station machine manipulates its timers using these shared variables.

The predicate action table for the generalized timer machine is shown in table I, and the state description is given in figure 1.

A. *State description*

The state machine at station i is specified in figure 3. The transitions for this state machine are outlined in table III. Each station machine has the following timers local to it: I-Timer, R-Timer, S-Timer, and B-Timer. The meanings of these timers are explained in table II. Each of these timer machines has two variables *st* and *val* as explained before. These are shared with

Abbreviation	Meaning
ACT	Active
EXP	Expired
INACT	Inactive
B-Timer	Backoff Timer
T-Timer	Timeout Timer
I-Timer	DIFS Timer
S-Timer	Short IFS Timer
R-Timer	Reservation Timer
STD	Value given by standard
BBusy	Busy during backoff

TABLE II
ABBREVIATIONS USED AND THEIR MEANINGS

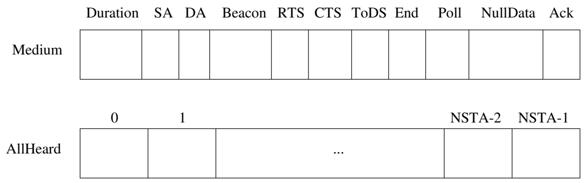

Fig. 2. Shared variables

Transition	Predicate	Action
DataReady	HaveData = true	B-Timer.val := rand()
WaitDIFS	Medium = 0 $\vee$ Backoff = false	I-Timer.st := ACT I-Timer.val := STD
Busy	Medium $\neq$ 0	I-Timer.st := INACT Backoff := true
DIFS-Over	I-Timer.st = EXP	B-Timer.st := ACT
BBusy	medium $\neq$ 0	B-Timer.st := INACT Backoff=true
Tx-Data	B-Timer.st = EXP $\vee$ (medium = 0 $\wedge$ Backoff=false)	Medium.DA : Address Medium.NullData := 0 T-Timer.val := STD T-Timer.st := ACT
Timeout	T-Timer.st = EXP	B-Timer.val := rand() Backoff := true
Rx-ACK	Medium.DA = i $\wedge$ Medium.ACK = 1	Backoff := false
Rx-Data	Medium.DA=i	S-Timer.val := STD S.Timer.st := ACT
Tx-ACK	S-Timer.st = EXP	medium.ACK := 1 Medium.DA := Address Medium.SA := i

TABLE III
PREDICATE-ACTION TABLE FOR BASIC DCF

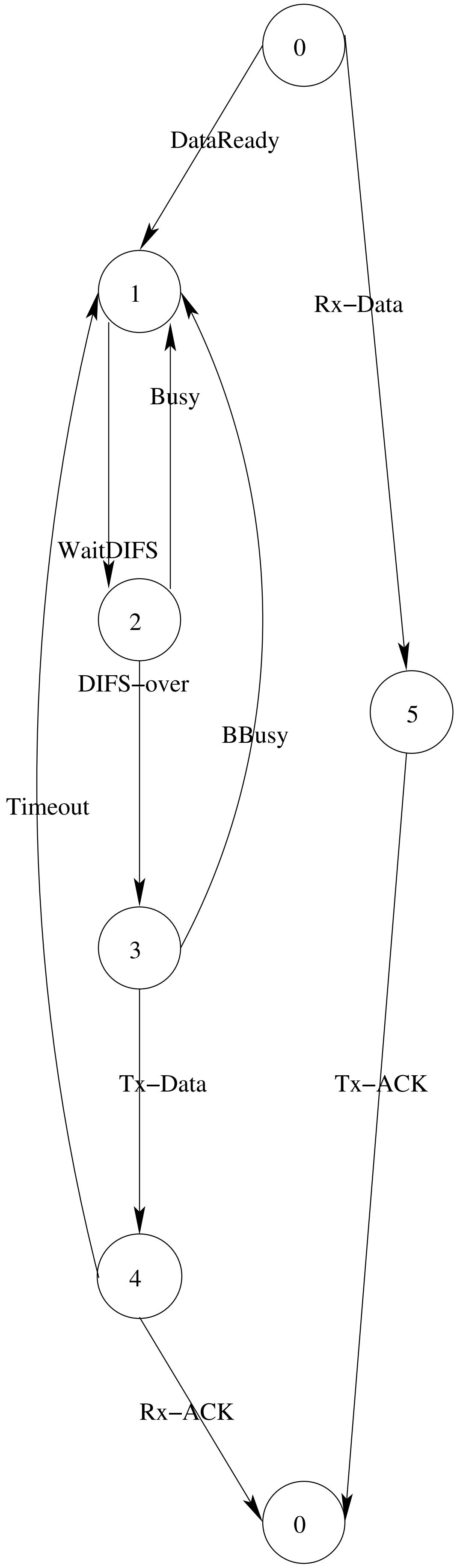

Fig. 3. State description of machine i

Variable	Range	Initial Value	Purpose
Backoff	boolean	false	True if backoff is enabled.
HaveData	boolean	false	True if data is to be sent. Set by external entity.
Medium		Empty	Represents the wireless medium

TABLE IV
VARIABLES IN THE DCF SPECIFICATION

the station machine which uses these timers. All variables are specified in table IV.

A brief description of the state machine follows. The machine starts in state 0. On reception of a data frame, the transition *Rx-Data* is enabled to get to state 5. Once in state 5, the machine performs the SIFS wait. Once this timer expires, *Tx-ACK* is enabled, the machine transmits the ACK frame, and gets to state 0. When a machine has a DATA frame to send, *DataReady* is enabled, and it gets to state 1. In state 1, the *WaitDIFS* is enabled if Backoff is false, which is true the first time it does carrier sensing, or if the medium is free and the machine gets to state 2. If the medium were busy any time, *Busy* is enabled, Backoff is set true, and the machine reaches state 1. If the medium were free for a DIFS duration, *DIFS-over* is enabled, and the machine reaches state 3. Once in state 3, if backoff was true, it performs the additional backoff wait. The backoff timer is reduced for the time the medium was free. If the medium becomes busy, the backoff timer is frozen, and the machine gets back to state 1 using *BBusy*- standing for Busy during Backoff. When the backoff timer becomes zero eventually, *Tx-Data* is enabled, and the machine gets to state 4. In state 4, the *Timeout* or the *Rx-ACK* transitions are enabled according to whether there is a timeout or an ACK reception respectively.

The following points should be noted:

- The number of retransmissions is limited. Once this limit is exceeded, the data frame is discarded, and the machine returns to state 0. We do not show this transition in the state machine.
- The reception of an ACK for a frame being currently retransmitted due to a timeout causes the system to declare the frame to be a success, and the machine returns to state 0. We also do not show these transitions in the state machine.
- The reception of a corrupt frame does not enable *Rx-data*. Therefore, the sending station eventually timesout and retransmits the frame for proper reception.

We now proceed to establish two lemmas, and subsequently prove theorems on freedom from deadlocks and non-executable transitions. We now proceed to establish two lemmas, and subsequently prove theorems on freedom from deadlocks and non-executable transitions.

Lemma 1: If a machine x gets to state 1, it gets to state 0 eventually.

Proof: The machine leaves state 0 to go to state 1 when it has a DATA frame to send. We first show that the machine gets to state 4 eventually. We consider two cases.

If the medium were free, and was so for a DIFS, the machine reaches state 3. In this case, the machine has observed the medium to be idle for a DIFS, and it goes ahead and transmits and gets to state 4, and we are done with the proof.

Suppose not. Some other machine is currently using the medium. x enables *backoff* to be true. All other machines in the neighbourhood of x also observe the medium to be busy. When the current transmission ceases, all machines wait for a DIFS before transmission. Therefore, the medium is free for a DIFS. In other words, machine x reaches 3 eventually. As x had observed the medium to be busy at some point, it has to perform the backoff procedure in state 3.

In state 3, let the backoff time chosen be δ. If the medium were free for some time ϵ, then when the medium becomes busy, the timer is *frozen* at $\delta - \epsilon$, and the state becomes 1. From our previous proof, the machine gets to state 3 eventually, but with the backoff timer $\delta - \epsilon$ i.e., a reduced backoff timer. In other words, the system eventually reaches state 3 with the backoff timer 0. When this happens, *Tx-Data* is enabled, and the machine gets to state 4.

Once the machine is in state 4, it waits for a pre-determined duration for the ACK. If the ACK is received within this duration, *Rx-ACK* is enabled and the machine reaches state 0. On the other hand, if it times out it goes back to state 1 by the transition *Timeout*. As before, the machine gets to state 4 again eventually. We note that the number of retransmissions is limited by the protocol to complete the proof, i.e, if there are more than the allowed number of retransmissions, the system gets back to state 0. ■

Lemma 2: If a machine x gets to state 5, it gets to state 0 eventually.

Proof: The machine gets to state 5, on receiving the DATA frame. Once it receives the data frame, the action for the *Rx-Data* frame starts the SIFS-timer. Once the SIFS-timer expires after an SIFS, the *Tx-ACK* is enabled, and the system gets to state 0. ■

Theorem 1: The CSMA/CA protocol as specified is free from deadlocks.

Proof: The state of the system at any instant of time can be specified in the form $(x_1, x_2, \ldots, x_n)$ where x_i represents the state of the machine i. Our proof is based on the fact that if a machine leaves the state 0, it returns to the state 0 eventually. As all machines start at state 0, they eventually get back to state 0. This has been established by the previous two lemmas. ■

Theorem 2: The CSMA/CA protocol as specified is free from non-executable transitions.

Proof: We see that all transitions are possibly enabled at some point of time i.e., there are none which can never be enabled, in the proofs for Lemmas 1 and 2. ■

We make the observation that there is no *liveness* in this protocol. If the random backoff timers of two machines choose the same value, the two machines transmit at the same time causing collision. It is perfectly possible that even after backing off subsequent to the collision, the machines transmit at the same time. The collisions could potentially continue indefinitely causing the protocol to lack liveness.

VI. Formal Specification of DCF with RTS/CTS: MACA

A. State description

In a manner identical to basic DCF, we model each station implementing MACA as a state machine. The state machine for station i is shown in figures 4, 5, and 6. The enabling predicates are shown in table V. The variables used are the same as in table IV. In addition, we have an R-Timer which counts when the medium has been reserved by another RTS, CTS, or data frame, which this station observes.

A brief description of the state transition diagram follows. Each machine starts at state 0. If the machine overhears an RTS/CTS/Data frame not meant for it, i.e., one that reserves the medium, the *Reserve* transition is enabled. It sets the *R-Timer* to the duration specified in the field of the frame, and enters state 8. Once this timer expires, this machine can now send data, and gets back to state 0. In addition, if it receives any further reservation information, it updates the waiting time and stays in state 8.

If a station needs to send data, it needs to send the RTS frame first. The station gets to state 1. It performs the DIFS wait in states 1 and 2. It invokes the backoff procedure in state 3 if necessary. Eventually, it transmits the RTS frame and waits for an acknowledging CTS frame in state 4. If it receives the CTS, it can move to state 5, where it performs the SIFS wait. Once the SIFS is over, the station transmits the DATA frame and moves to state 9. In state 9, the machine waits for an ACK. The transitions *Timeout* and *Rx-ACK* are enabled if there is a timeout or an ACK respectively. In state 4, if it does not receive the CTS, it times out and reaches state 1.

When a station receives data, *Rx-Data* is enabled, and the machine gets to state 7. In state 7, it waits for an SIFS duration. At the end of the SIFS, *Tx-ACK* is enabled, and the machine gets to state 0. Similarly, when a machine receives an RTS meant for itself, it waits for an SIFS and proceeds to transmit the CTS, and gets back to state 0.

The number of allowed timeouts waiting for a CTS frame or the ACK frame is limited, and if this number is exceeded, the machine gets back to state 0. We do not show these transitions explicitly.

We now proceed to establish the following lemmas, and prove theorems on safety of the MACA protocol.

Lemma 3: If a machine x gets to state 6, it gets to state 0 eventually.

Proof: If the machine receives the RTS, it needs to wait for an SIFS. This is done in the state 6. Once this SIFS duration is over, the machine's *Tx-CTS* is enabled, and the machine gets to state 0.

■

Lemma 4: If a machine x gets to state 1, it gets to state 5 or state 0 eventually.

Proof: When a machine needs to send data, it needs to send the RTS frame for the data. The proof is identical to the proof for Lemma 1 except for the fact if the machine times out repeatedly waiting for the CTS it reaches state 0. Otherwise, the machine reaches state 5.

■

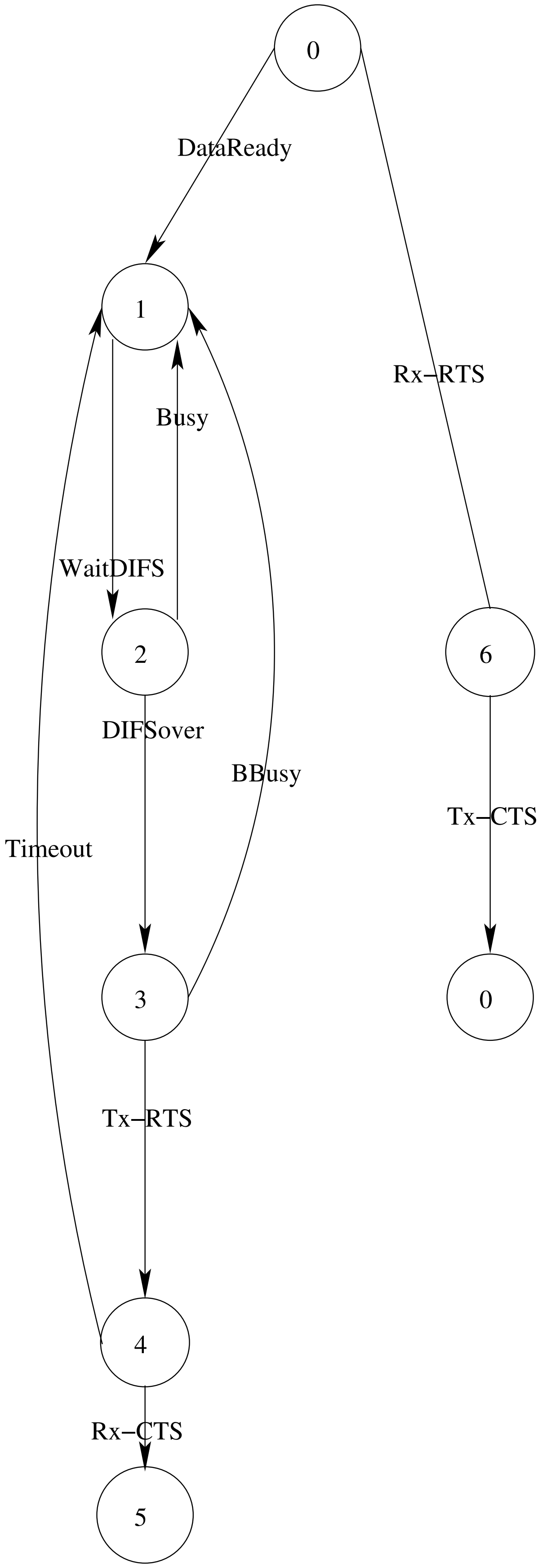

Fig. 4. State machine for station i contd.

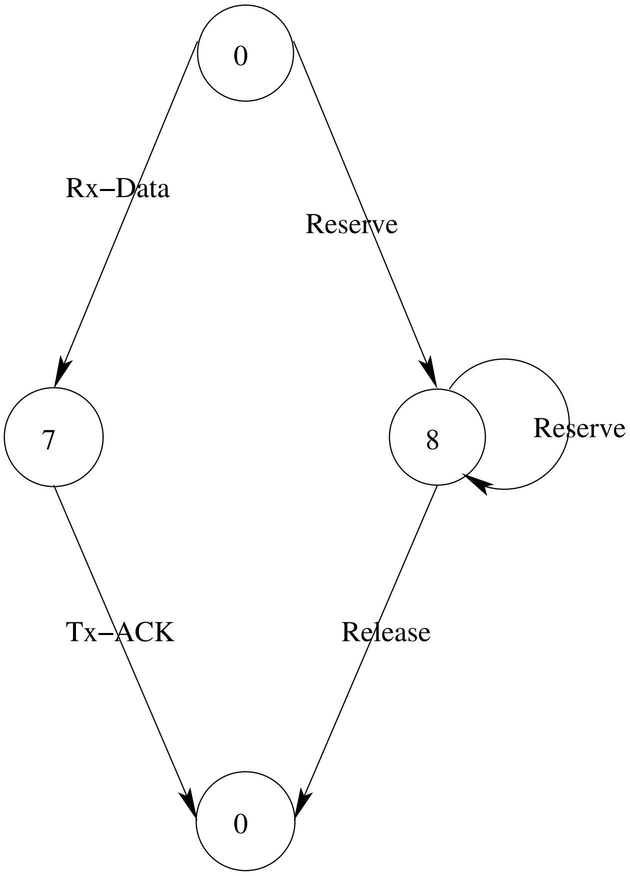

Fig. 5. State machine for station i contd.

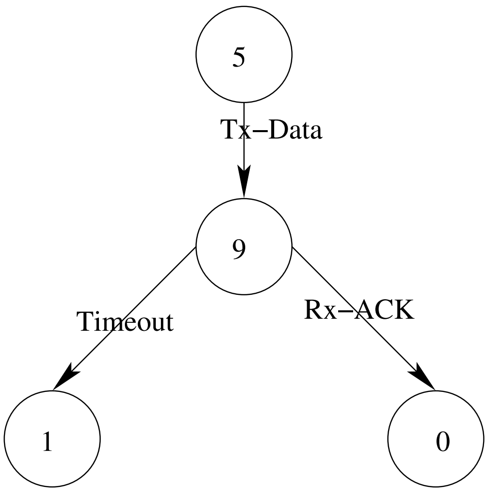

Fig. 6. State machine for station i

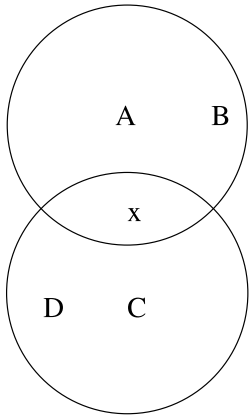

Fig. 7. Potential Livelock in MACA

Transition	Predicate	Action
DataReady	HaveData = true	B-Timer.val := rand() Backoff:=false
WaitDIFS	Medium = 0 $\vee$ Backoff=false	I-Timer.st:=ACT I-Timer.val := STD
Busy	Medium $\neq$ 0	I-Timer.st:=INACT
DIFS-Over	I-Timer.st = EXP	B-Timer.st:=ACT
BBusy	Medium $\neq$ 0	B-Timer.st :=INACT Backoff := true
Tx-RTS	B-Timer.st = EXP $\wedge$ $\vee$ (Medium = 0 $\wedge$ Backoff=false)	Medium.RTS := 1 Medium.DA := Address T-Timer.val := STD T-Timer.st := ACT
Timeout	T-Timer = EXP	B-Timer.val := rand() Backoff:=true
Rx-CTS	Medium.DA = i $\wedge$ Medium.CTS = 1	S-Timer.st:=ACT
Rx-RTS	Medium.DA = i $\wedge$ Medium.RTS=1	S-Timer.st:=ACT
Tx-CTS	S-Timer.st=EXP	Medium.CTS := 1 Medium.DA = Address
Rx-Data	Medium.DA = i	S-Timer.st:=ACT
Tx-ACK	S-Timer.st=EXP	Medium.ACK := 1 Medium.DA := Address
Reserve	Medium.DA != i $\wedge$ (Medium.RTS=1 $\vee$ Medium.CTS =1 $\vee$ Medium.NullData=0)	R-timer.st:= ACT R-Timer.val := max(R-Timer.val, Medium.Duration)
Release	R-Timer.st=EXP	
Tx-Data	S-Timer.st=EXP	Medium.DA := Address Medium.NullData := 0 T-Timer:=ACT
Timeout	T-Timer.st=EXP	backoff=true
Rx-ACK	Medium.DA = i $\wedge$ Medium.ACK=1	

TABLE V
PREDICATE-ACTION TABLE FOR DCF WITH RTS/CTS: MACA

Lemma 5: If a machine x gets to state 5, it gets to state 0 eventually.

When the machine gets to state 5, it needs to send a data frame. It sends it after waiting for an SIFS by enabling *Tx-Data*. If it is ACKed before the timeout, it reaches state 0. If it times out, it reaches state 1, and by the previous lemma, gets back to state 5. If it receives an ACK, it reaches state 0, else the process repeats. We complete our proof by noting that the number of retransmissions is limited, and the machine eventually gets back to state 0 even in this case.

Lemma 6: The MACA protocol has a potential livelock in state 8.

Proof: Consider figure 7. Station x which can hear all of the stations A,B,C, and D. A and B are out of the hearing range of C and D, and vice-versa.

Suppose A needs to talk to B, and it sends an RTS with a time duration δ. When x receives this RTS at time t, it defers its transmission and marks the NAV as busy between t to $t + \delta$.

Suppose C needs to transfer data at $t + \delta - \epsilon$, where ϵ is the IFS wait. C doesn't hear the medium to be busy at this time, and it waits for a IFS ϵ time. It transmits the RTS at time $t + \delta$ precisely forcing x to wait for another reservation.

When C and D are done talking, A and B could start all over again in the exactly same manner forcing x to wait indefinitely. This means that x could be livelocked in state 8.

While in practice, it will be unlikely stations will send at the chosen times exactly causing x to be starved, the protocol *does not guarantee* this causing the MACA protocol to have a potential livelock. ∎

Lemma 7: If a machine gets to state 7, it gets to state 0 eventually.

Proof: Once the machine receives a DATA frame, it enters state 7. In state 7, it waits for SIFS. Once the SIFS is over, it transmits the ACK and gets back to state 0. ∎

Theorem 3: The MACA protocol as specified is free from deadlocks modulo lemma 6.

Proof: The state of the system at any instant of time can be specified in the form $(x_1, x_2, \ldots, x_n)$ where x_i represents the state of the machine i. Our proof is based on the fact that if a machine leaves the state 0, it returns to the state 0 eventually except if they get livelocked in state 8. As all machines start at state 0, they eventually get back to state 0 provided we assume that the machines don't get livelocked in state 8. This has been established by the previous two lemmas. ∎

Theorem 4: The MACA protocol as specified is free from any non-executable transitions.

Proof: Our proof is based on the fact that all transitions are enabled at some point in time as seen in the proofs of the previous two lemmas.

∎

VII. Discussion

In this section, we seek to understand the results we have obtained. Firstly, the freedom from deadlocks is guaranteed by the wait-state basically. Everybody who sees the medium, waits till the medium is free. In addition, when the medium is free the B-timer, i.e., the backoff timer is reduced but not reset. This means, that everybody gets a chance eventually. If a machine has waited for some time, it is counted as credit, and it reduces the waiting time finally.

The communication paradigm followed in the design of the 802.11 DCF is *smart sender - dumb receiver*. In other words, the responsibility of ensuring that a proposed communication took place resides with the sender. This guarantees that there is no deadlock when a machine receives something. It simply waits for a short time (SIFS), and transmits an ACK or CTS as required without worrying about whether the medium is free. If the medium were not free, it is upto the sender to retransmit the RTS or the DATA frame as appropriate.

We observed the fact that the MACA protocol has a potential livelock in the ad-hoc mode. Intuitively, one can see that any random scheme does not ensure a fair share of resources. While we can say a station would get a chance with high probability, it is not a guarantee written in stone. For such tight guarantees at the MAC layer, one would need some sort of token based approach like FDDI.

Finally, we observe the fact that while we demonstrated safety properties of the protocol, there were no guarantees on liveness, i.e., something happening within a finite duration of time. Again, we blame randomness for this. For instance, two machines could perform the backoff and retransmit at exactly the same time causing collision. This could go on indefinitely. While we don't expect this to happen with high probability - considering the fact that the backoff times are randomly chosen from an ensemble, *there are no guarantees*.

VIII. Conclusions and future work

A formal description method called the *systems of communicating machines* has been used to specify and verify the 802.11 DCF protocol in machines working in the ad-hoc mode of operation.

The analysis demonstrated that the CSMA/CA protocol is free from deadlocks and non-executable transitions. We were able to point out a potential livelock situation in the DCF protocol with the RTS/CTS exchanges. While we concede that this is not always bound to happen, the DCF protocol with RTS/CTS exchanges does not guarantee fairness.

Specifying the protocol in this model has some advantages over other formal description techniques. In the 802.11 standard, the MAC is specified using a High Level Protocol Description language. While this does specify the protocol in all gory detail, it is not amenable to verification techniques, and more importantly, provides no intuition behind the operation of the protocol whatsoever. SCMs offer formalism as well as programming language style syntax. More importantly, it is easy to visualize the operation of the protocol. This leads us directly to postulate and prove invariance properties of the protocol.

An idea which can be pursued further is modelling mobility explicitly using SCMs. Currently, we model mobility implicity, i.e. a station hears another station if it is near enough. In other words, it is left unspecified when exactly a station is within the hearing distance of another station.

Just as we fake time using timers, we can fake space using co-ordinates in the SCM model. The position of each station could be defined by two co-ordinates. The distance between any two stations is defined as the euclidean distance between them.

The hearing range of a station is modelled as a circle with center at its current position in space, and radius determined by the actual physical characteristics of the medium, i.e, at this frequency band, how long can the signal traverse in space without being completely damped. Any station receives a frame broadcast by another station if, and only if, it is within the hearing range of another station. We propose to study this approach in greater detail.

REFERENCES

[1] Institute of Electrical and Electronics Engineers, Inc., *ANSI/IEEE Standard 802.11*, 1999

[2] P.Karn, MACA: A new channel access method for packet radio. *Proceedings of the ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, September 1990. `http://people.qualcomm.com/karn/papers/maca.html`

[3] G.M.Lundy and I.F.Akyildiz, *Specification and analysis of the FDDI MAC protocol using systems of communicating machines*, Computer Communications, Vol. 15, No. 5, 1992, pp285-294

[4] G.M.Lundy and R.E.Miller, *Analyzing a CSMA/CD protocol through a system of communicating finite state machines specification*, NASA Center for Excellence in Space Data and Information Science, TR-90-01, Greenbelt, MD, Jan 1990.

[5] G.M.Lundy and R.E.Miller, *Analyzing a CSMA/CD Protocol through a system of communicating machnes specification*, IEEE Transactions on Communications, Vol. 41, No. 3, March 1993

[6] G.M.Lundy. *"Systems of Communicating Machnes: A model for communication protocols"*, Ph.D. Dissertation, School of Information and Computing Science, Georgia Inst. Tech., Atlanta, GA, 1988

[7] M.A.Youssef and R.E. Miller, *Analyzing the Point Coordination Function of the IEEE 802.11 WLAN Protocol using a systems of communicating machines specificatios"*,Technical Report, UMIACS-TR 2002-36 and CS-TR 4357, Department of Computer Science, University of Maryland, College Park, May 2002