IEEE 802.11 Wireless LAN Model Documentation
April 8, 1998

Rusty O. Baldwin
Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0111

Introduction

This document is the documentation for the IEEE 802.11 simulation model. Section 1 contains a discussion of the simulation tool. The next section, 2, presents the simulation model itself. This section highlights the structure of the model and documents the behavior using the Specification and Description Language (SDL-92) design language which was used in the IEEE 802.11 standard. A description a this language can be found in [EHS97]. The next section describes the simulation parameters which can be varied within the model. Section 4 discusses the validation of the model and the document is concluded with a summary in Section 5. A concise overview of the model and SDL symbols is contained in the paper "Implementation of an IEEE 802.11 Wireless LAN Model using OPNET ™" by Rusty O. Baldwin, Nathaniel J. Davis IV, and Scott F. Midkiff which should accompany this document.

1 Simulation Tool

This model used the communication network simulator OPNET™, version 3.0.B by MIL3, Inc. [MIL97]. Simulations in OPNET are organized in a hierarchical structure of models consisting of (from the highest to lowest level) network models, node models, process models, and parameter models. Network models essentially describe the physical location of nodes and how they are connected (i.e., radio, bus, ring, etc.). Nodes are the stations in the network. Node models describe connections between the process models. Processes specify behavior and parameter models define the data structures used by processes for inter-process communication. Parameter models also define the structure of the packets transmitted on the network itself. At a given level in the hierarchy, a model is defined using the models at the next lower level. For example, a network model consist of a set of node models arranged and connected in a particular way. Likewise, a node model consists of a number of process models connected in a given way. These different models can be thought of as objects, each with a set of attributes that can be modified. Thus, OPNET can be described as somewhat object-oriented in its approach to modeling components within the simulation program. Data from the simulation is collected by placing statistic "probes" at points of interest within the models. Data collected can be analyzed using the analysis tool provided or exported to be analyzed in external programs.

2 Wireless LAN Simulation Model

This section will document the IEEE 802.11 simulation model. It will begin at the node level and proceed down the hierarchy to processes and parameters. The graphical portion of the SDL-92 specification language used in the IEEE 802.11 standard [IEE97] will also be used to document this simulation model. While the specification contained in [IEE97] is a complete, valid SDL specification, the purpose in this section is to document the behavior of the OPNET process models. Therefore, while elements of the SDL graphical language are used in this section, it is not a complete SDL specification of the OPNET model.

The simulation model implements the distributed coordination function (DCF) of IEEE 802.11. Stations (nodes) within the model form an ad-hoc network (i.e., an independent basic service set (IBSS) in IEEE 802.11 terms). IEEE 802.11 functions not in the model include: encryption, authentication, power-save mode functions, fragmentation/de-fragmentation, and the point coordination function (PCF).

2.1 Node Model

The highest level SDL block is the System Station. This block is shown in Figure 1 below and contains other blocks which specify functions within System Station. The signal names match the state transition condition name or state executive statement name used in the OPNET process models. The corresponding node level OPNET implementation is shown enclosed in the dashed line box in Figure 2; the solid line boxes are the MAC Service Access Point, and the PHYSICAL Service Access Point. These functions are external to the System Station and are not defined by IEEE 802.11. The signals in Figure 1 are generally implemented by interrupts in OPNET and do not appear explicitly in Figure 2. The solid and dotted lines with arrows in Figure 2 represent packet flow and statistic probes within the System Station respectively.

Moving down one level in the hierarchy, the blocks within Figure 1 (e.g., Protocol_Control_STA, Transmission, and Reception) are defined in terms of process

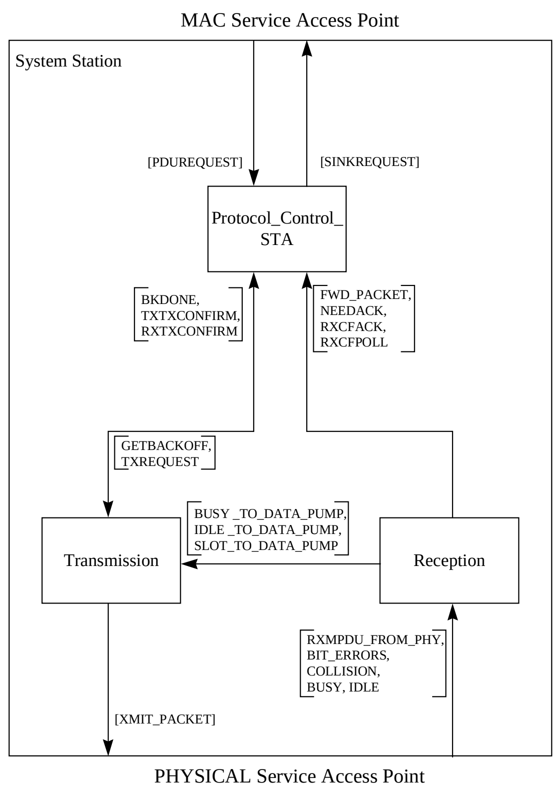

Figure 1: SDL - System Station

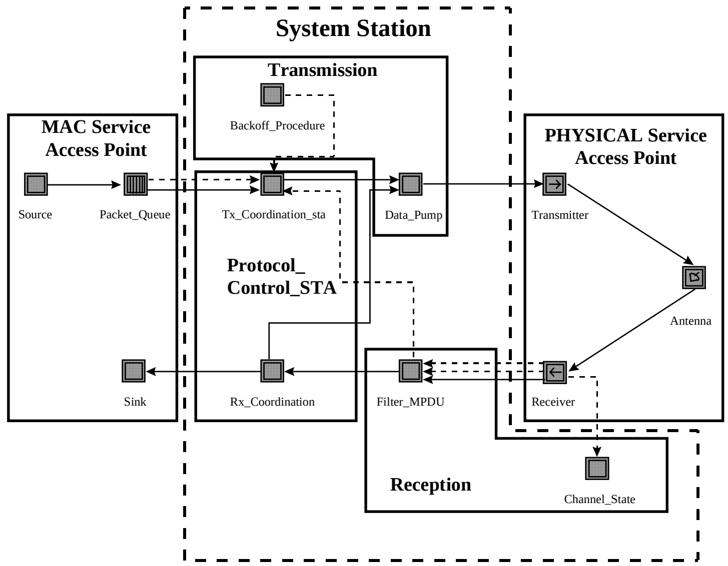

Figure 2: OPNET Node - System Station

models. Figures 3, 4, 5 show the Protocol_Control_STA, Transmission, Reception blocks respectively.

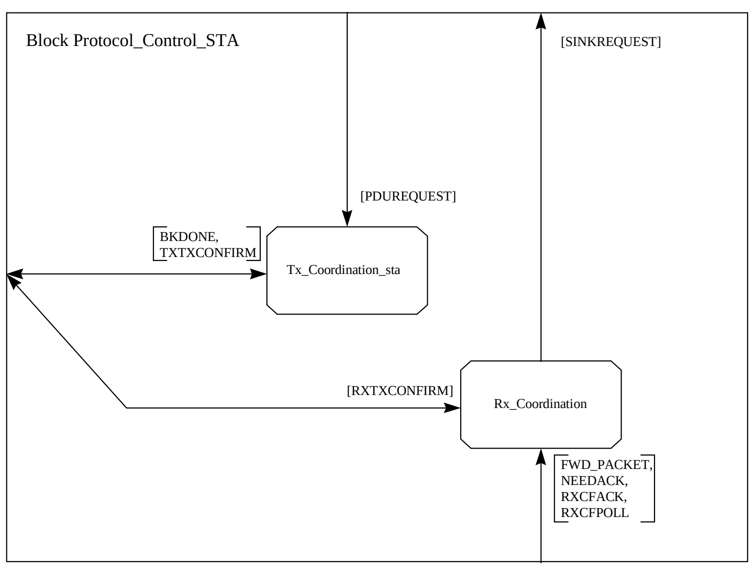

Figure 3: SDL - Block Protocol_Control_STA

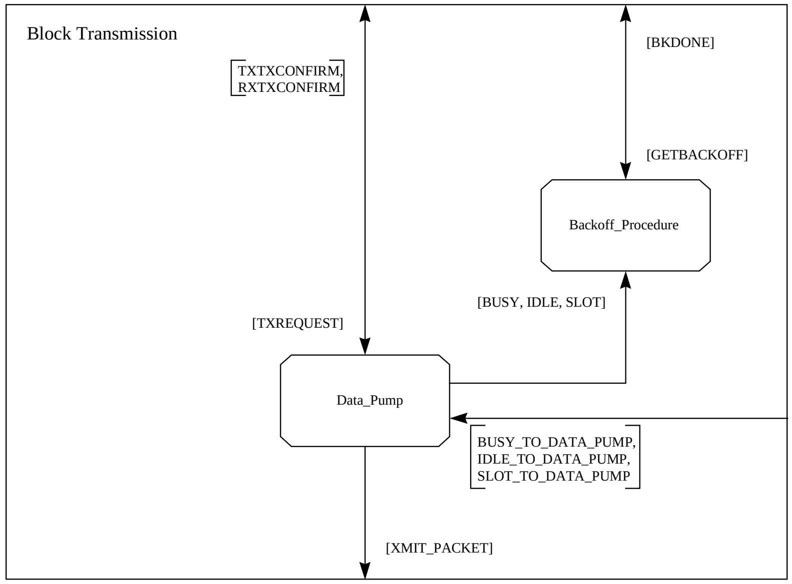

Figure 4: SDL - Block Transmission

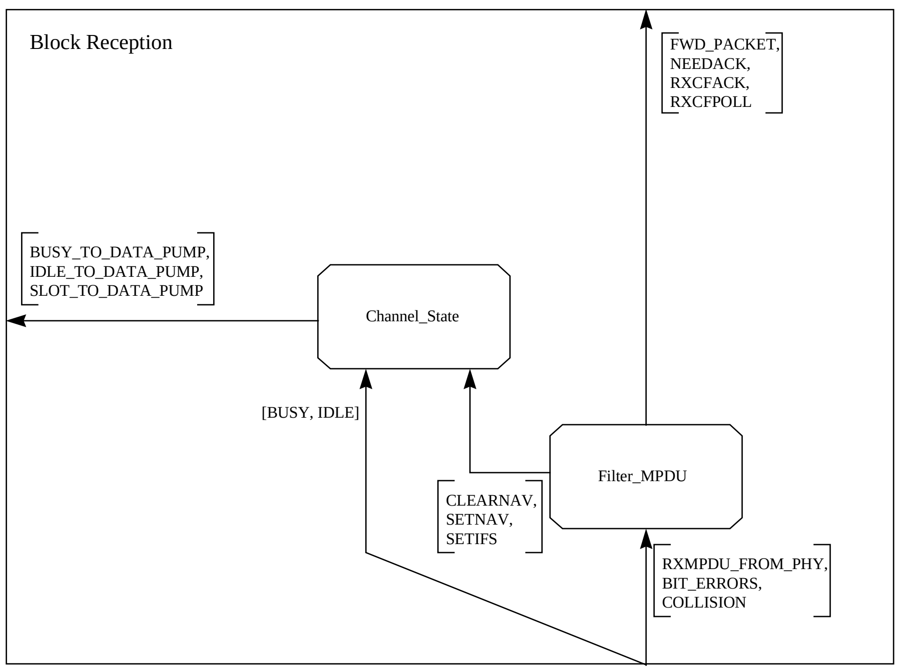

Figure 5: SDL - Block Reception

2.2 Process Models

Process models describe the behavior of process objects and are essentially finite state machines. The following sections describe the process models in the IEEE 802.11 simulation model.

2.2.1 Source

The Source process model is not part of the IEEE 802.11 specification. It generates the packets that the station transmits. Source can generate three classes of data packets: hard real-time (i.e., packets with deadlines that cannot be missed), soft real-time (i.e., packets with deadlines that can be met within a certain tolerance), and normal data packets with no deadlines. Each class of data packets has up to three independent input streams that can be specified to simulate different applications running on the station. The arrival, service, and packet deadline distributions can be any of the predefined distributions supported by OPNET. Figure 6 shows the OPNET implementation of the Source process model. The circles are states that the process can be in. Source has three states: Start, Create Packet, and idle. Within these states, processing, in the form of C language statements, is performed. These C language statements are not shown. The large solid arrow indicates the state the process starts in. Typically, any needed initialization is performed in this state. The shaded circles indicate that any processing associated with the state is performed and then the state is exited. OPNET terms this a "forced" state. A process remains in an unshaded state until a transition condition

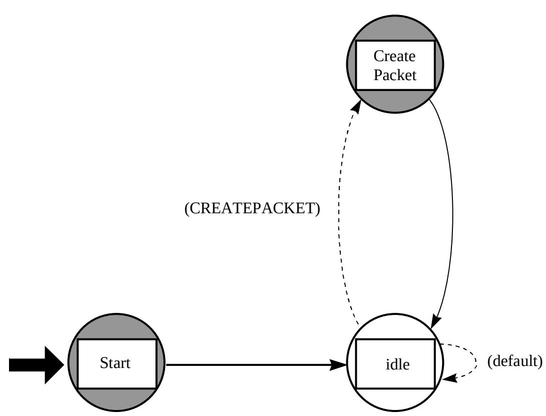

Figure 6: Source Process Model

becomes true. Transitions that occur in response to a particular condition are shown by a dotted line with an arrow. The condition that triggers the transition is shown in parenthesis beside the line (e.g., (CREATEPACKET) in Figure 6). Solid lines with arrows show unconditional transitions.

In the process Source, initialization occurs in the Start state. This initialization includes setting up the interrupts for the arriving packets and other OPNET specific initializations. After this is done, Source goes to the idle state. The only condition which will cause Source to leave the idle state is CREATEPACKET. All other conditions cause idle to be exited and re-entered via the default transition. The CREATEPACKET condition is implemented in C and becomes true when any interrupt to create a packet occurs. When CREATEPACKET becomes true, the process goes to the Create Packet state. In this state, the interrupt source is determined, the particular class of packet indicated by the interrupt (i.e., hard, soft, or data) is created and sent to the process Packet_Queue. Source then unconditionally returns to the idle state.

2.2.2 Packet_Queue

Packet_Queue receives packets from Source and sends packets, upon request, to the process Tx_Coordination_sta. Figure 7 is the process model for Packet_Queue. Packet_Queue, as with Source, is not part of the IEEE 802.11 specification.

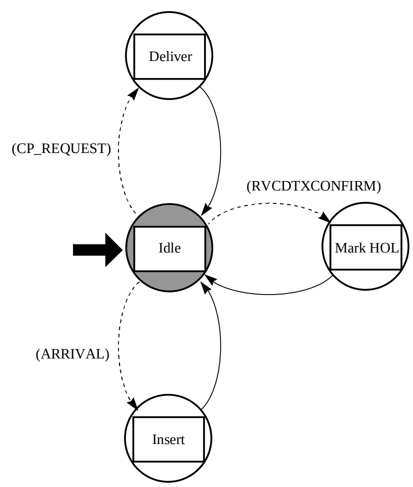

Figure 7: Packet_Queue Process Model

Packet_Queue begins in the Idle state. If CP_REQUEST (i.e., contention period packet request) become true, it goes to the Deliver state and delivers the next eligible packet. The delivery order is by packet class; first, hard real-time packets, next, soft real-time packets, and finally, data packets. The queue discipline is FCFS. Other queueing disciplines will be added at a later date. ARRIVAL is true when a packet arrives from Source. The arriving packet is placed in the appropriate queue (i.e., hard, soft, or data). After a packet has been transmitted, RCVDTXCONFIRM becomes true. Packet_Queue transitions to the Mark HOL state and marks the next packet to be selected for transmission with the current time. This is done for statistical purposes only.

2.2.3 Sink

The Sink process receives packets from the Rx_Coordination process. Currently, its only function is to destroy the packet. Its process model is not shown.

2.2.4 Transmitter, Receiver, and Antenna

The last process models which are not part of the IEEE 802.11 specification to be discussed are the transmitter, receiver, and antenna. The simulation model uses the default transmitter, receiver, and antennas available in the Node Editor. Their process models are not accessible to the user and are not shown.

2.2.5 Backoff_Procedure

This process implements the backoff function of IEEE 802.11 and is shown in Figure 8. After Start, the process is in the No_Backoff state. This means that either the backoff counter is not active or has reached zero. It stays in this state until it gets a request to choose another backoff count (i.e., RCVDBACKOFF). The condition associated with this transition is different from what has been seen before. It has a condition (i.e., (RCVDBACKOFF)) and what is known as an executive (i.e., the statement following the slash). The executive is a C language statement, in this case a function, that is executed when the condition is true and prior to arriving in the destination state. The executive get_slotCnt() chooses the initial backoff value. The process then enters the Channel_Busy state. It stays in this state until the channel is detected idle, an idle slot is detected or a cancel signal is received (which means the packet to be transmitted will not be transmitted after all). If a cancel is received (i.e., RCVDCANCEL is true), the process returns to the No_Backoff state. If IDLE_SLOT is true the process goes to the state Channel_Idle. In this state, the counter is decremented once for every slot (i.e., SLOT_COUNT_NOT_ZERO/decrement_slot_count()). If the channel becomes busy, the process returns to Channel_Busy. When the backoff count reaches zero in the Channel_Idle state, the BkDone signal is sent (i.e., SLOT_COUNT_ZERO_AND_NOT_BUSY/send_bkdone_minus_one()). The graphical SDL description of this part of the simulation model is shown in Figure 9. This SDL description defines the behavior of the process and shows the computations that the executive statements and process states implement.

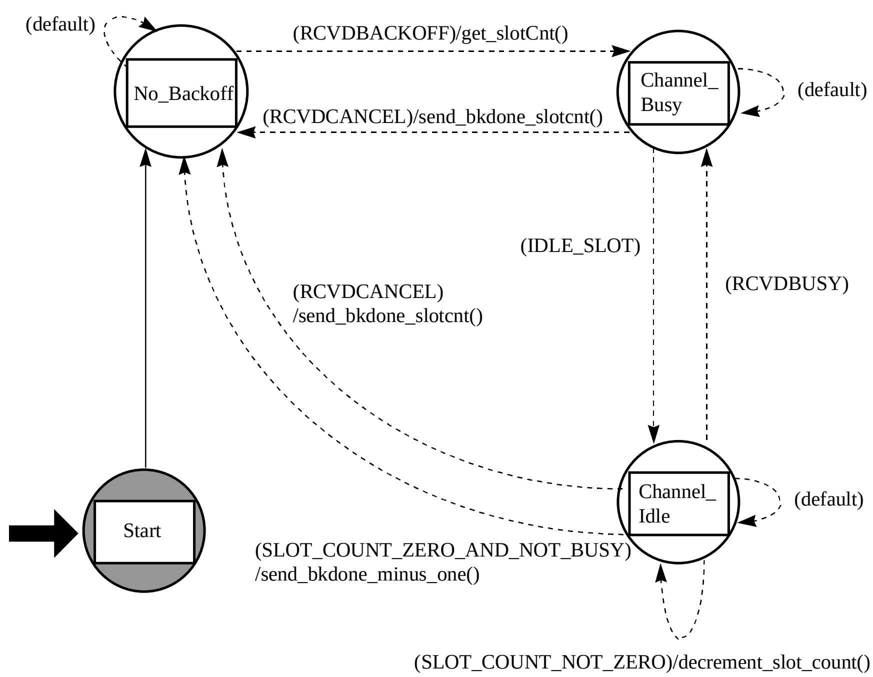

Figure 8: Backoff_Procedure Process Model

2.2.6 Data_Pump

Data_Pump (Figure 10) is the process that receives the packet to be transmitted and places it onto the channel. The process begins, as usual, at Start and immediately proceeds to the TX_Idle state. If the process receives a TXREQUEST, it obtains the packet to transmit, sends a busy signal to the station and goes to Wait_TxStart. It will stay in this state until an idle or a slot has been detected, whereupon it will go to the Send_Frame state and transmit the frame. After transmitting the frame, it returns to the TX_Idle state. A similar procedure occurs from the TX_Idle state when the ACKREQUEST condition is true. In this case, however, the process does not need to wait for an idle or a slot to be detected since sending an ACK implies that this station alone needs to respond. Therefore, there is no possibility of a collision occurring. The SDL description of Data_Pump behavior is shown in Figure 11 below.

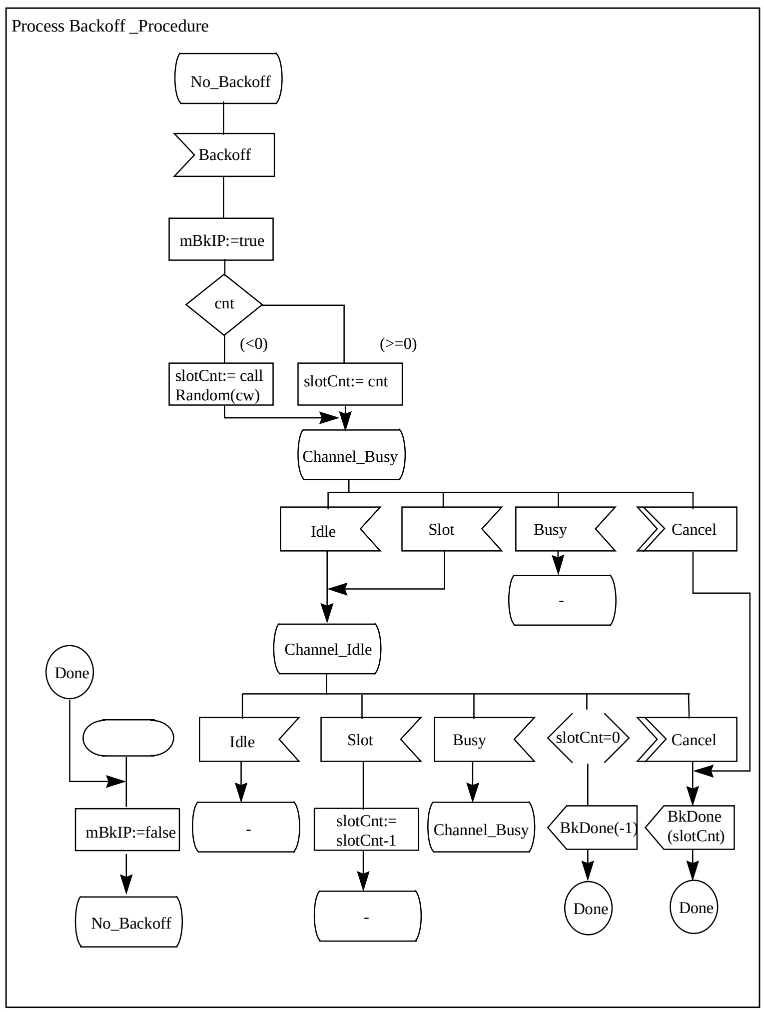

Figure 9: Backoff_Procedure - SDL Description

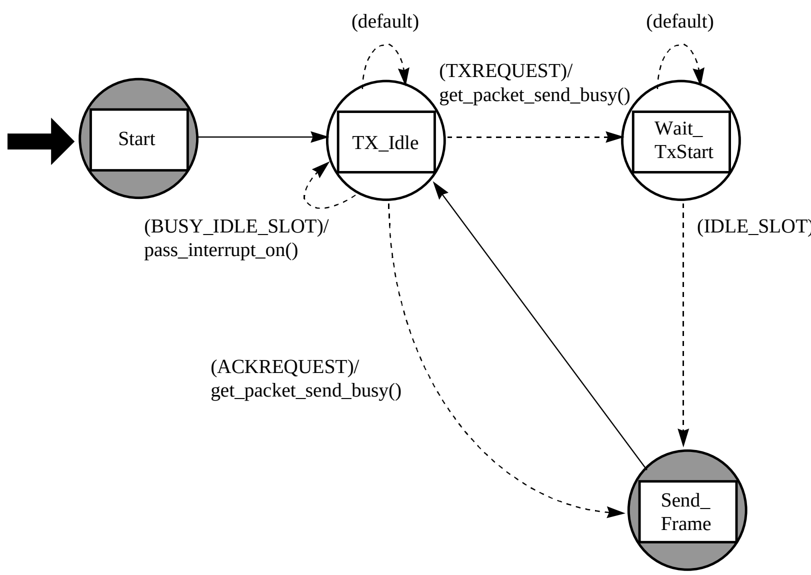

Figure 10: Data_Pump Process Model

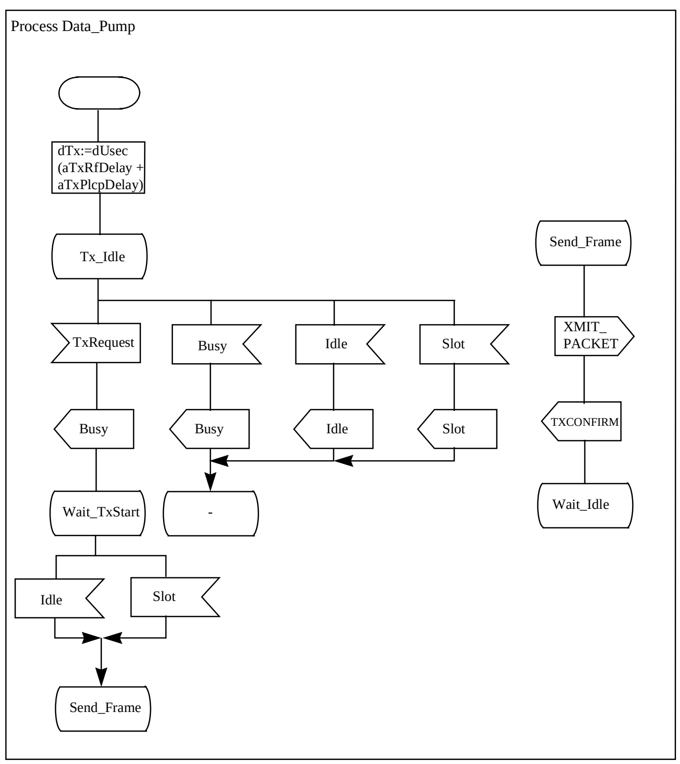

Figure 11: Data_Pump - SDL Description

2.2.7 Filter_MPDU

This process determines whether a received packet is bound for the station that received it. In the simulation model, its function is greatly simplified since the process does not need to handle the point coordination function (PCF) control packets, authentication, or multi-cast packets and the like, which have not been implemented. The Filter_MPDU process model is shown in Figure 12. After beginning in the Start state, the process proceeds to Filter_Idle. In this state, an incoming packet that is corrupt due to a collision or to bit errors will cause the process to set the bad packet flag. The indication of a collision or a packet corruption is received prior to the packet being received, therefore, the bad packet flag indicates the next packet received is corrupt. When the packet arrives, the process transitions to the Process Packet state. In this state the packet is discarded if it is corrupt. If it is not corrupt, the process determines whether the packet is bound for this station. If so, it is sent on. If not, the Network Allocation Vector (NAV) contained in the packet (e.g., the amount of time the channel will be in use due to this transmission) is obtained, and then the packet is discarded. The process model is quite simple since most of the processing is contained within the Process Packet state. The SDL description of the process are shown in Figures 13 and 14. The cache referred to in the SDL description is a cache of packet identifiers to permit duplicate packet filtering. This allows the station to detect and discard packets that may have been resent due to a lost ACK.

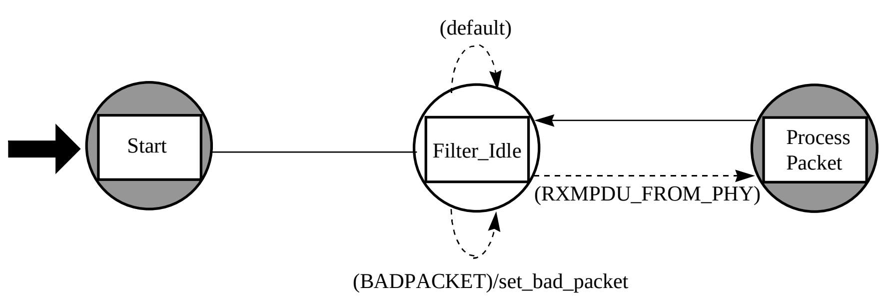

Figure 12: Filter_MPDU Process Model

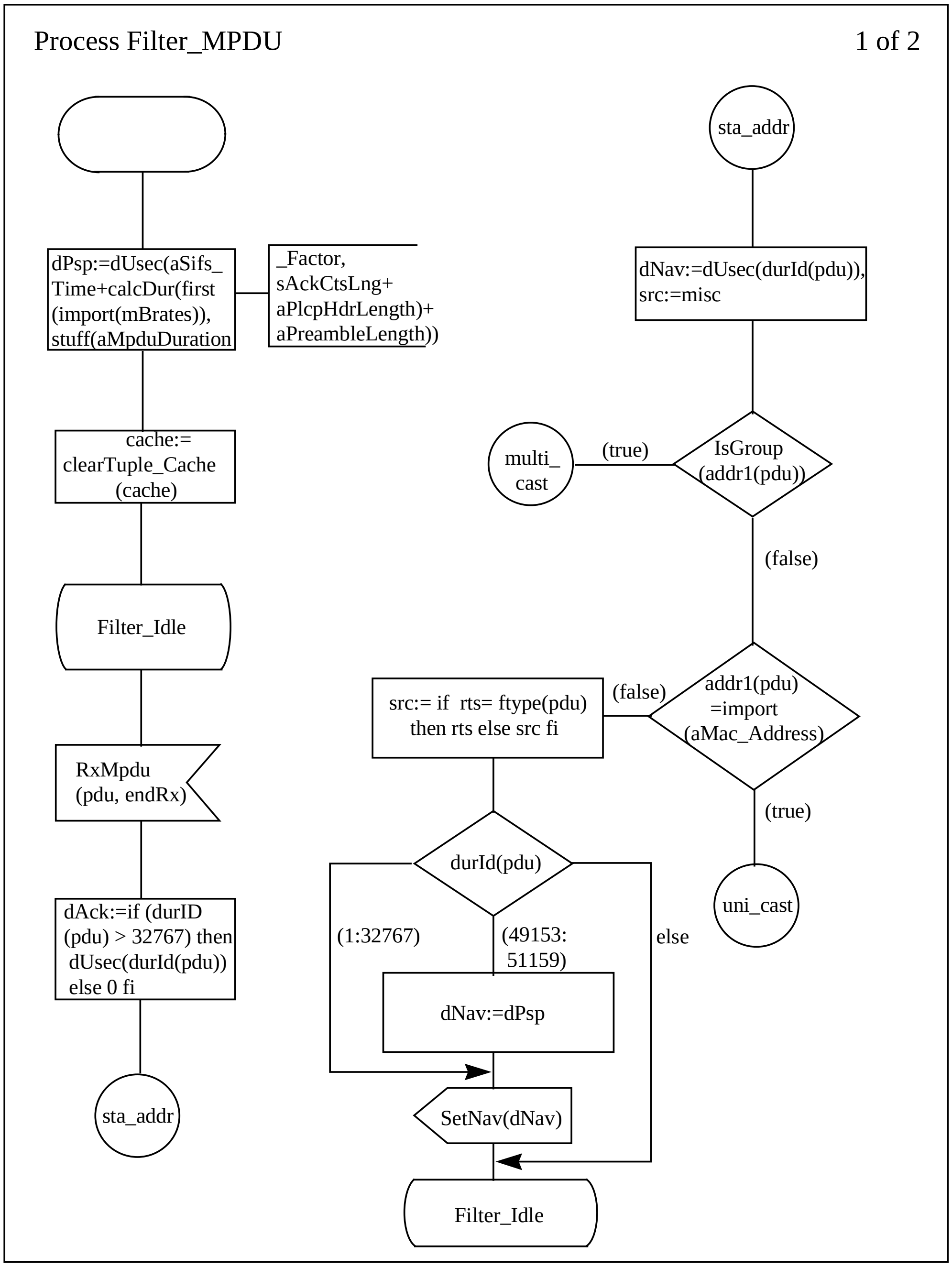

Figure 13: Filter_MPDU - SDL Description (1 of 2)

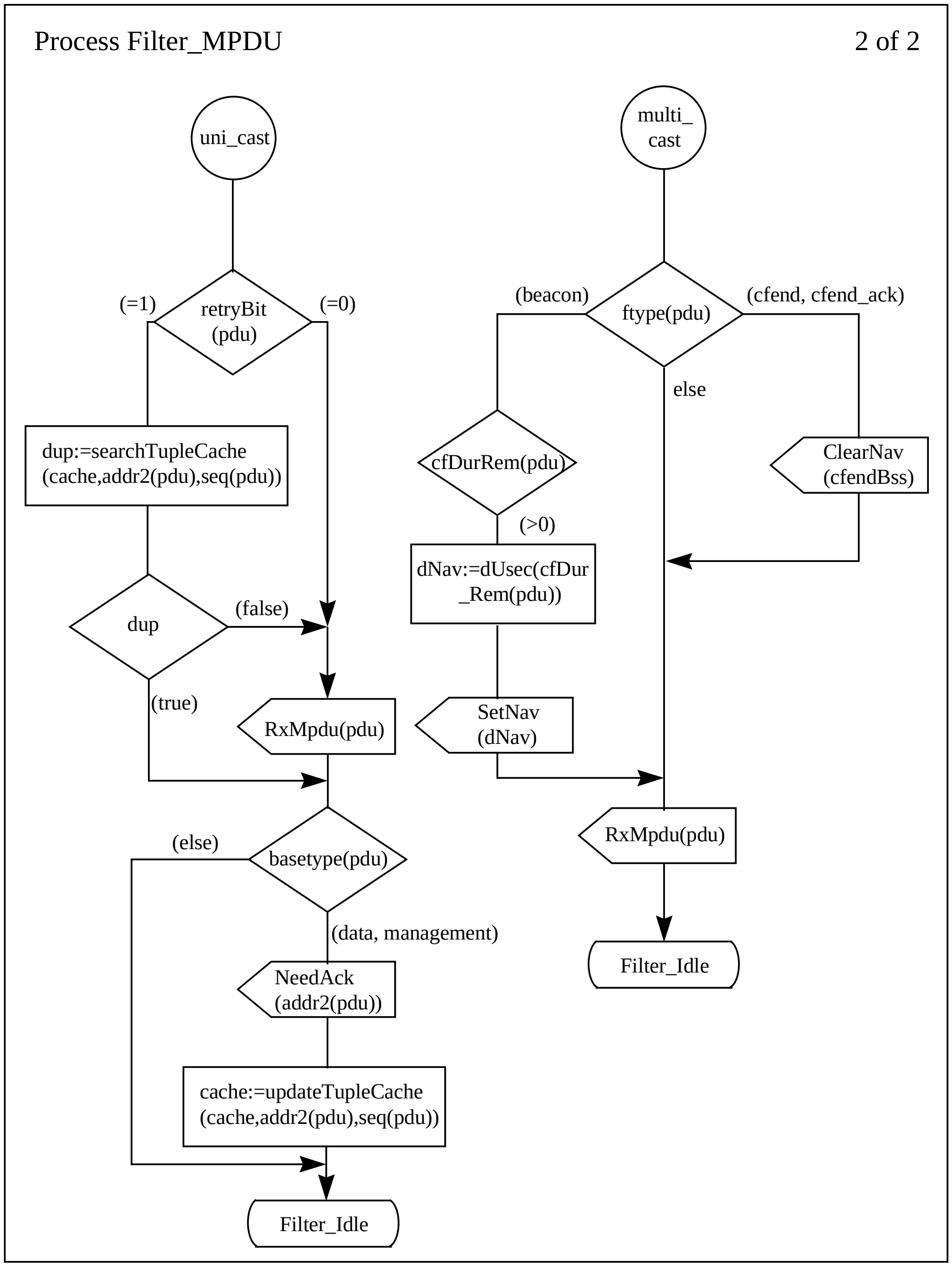

Figure 14: Filter_MPDU - SDL Description (2 of 2)

2.2.8 Channel_State

The state names in the Channel_State process (Figure 15) reflect the physical and virtual busy channel detection capability of an IEEE 802.11 station. The physical busy channel detection capability uses a standard carrier sensing (CS) process. The virtual busy channel detection capability uses a network allocation vector (NAV). The NAV is a value contained in an incoming packet which indicates the amount of time the channel will be needed to complete the current transmission (i.e., transmit an acknowledgment in response to a received packet). Stations in the network will not initiate a transmission for the duration of a NAV. This ensures that the ACK for the current packet does not collide with a new packet transmission. Both the CS and the NAV must indicate an idle channel for a station to initiate a transmission. The process will be in state Cs_Nav when the channel is both physically (CS) and virtually (NAV) busy. The process will be in Cs_noNav when the channel is physically busy but not virtually busy, and so on.

As always, the process model begins in the Start state. From there is goes to the Cs_noNav state. There are several transition conditions which are common to most

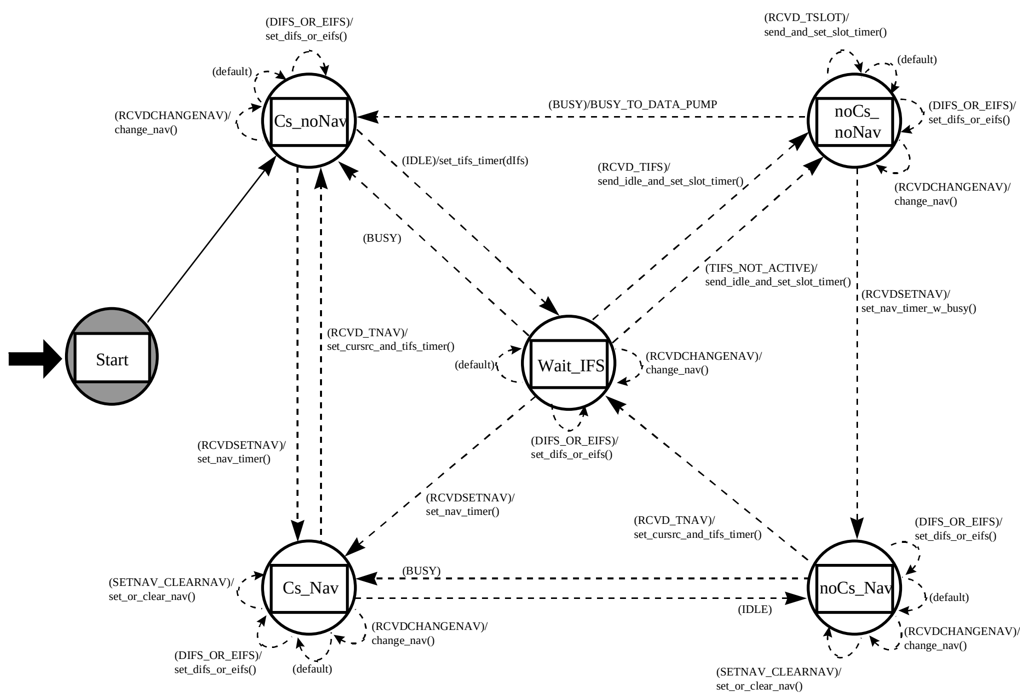

Figure 15: Channel_State Process Model

states. Those common transition conditions are: RCVDCHANGENAV, DIFS_OR_EIFS, and SETNAV_CLEARNAV. The condition RCVDCHANGENAV becomes true after the Filter_MPDU process decodes a packet containing a NAV and determines that the current NAV needs to be changed. Channel_State will update its NAV and return to whatever state it was in prior to the condition becoming true. When the condition DIFS_OR_EIFS is true, Channel_State will update the value used for the inter-frame space (IFS) and then return to the state it was in prior to DIFS_OR_EIFS becoming true. The IFS is the amount of time the station waits after the channel is both physically and virtually idle before actually declaring the channel idle. The DIFS value is the smaller value and is used as the IFS when the packet cyclic redundancy code (CRC) is verified as good. If the CRC check fails, then the longer EIFS is used for the IFS. The condition SETNAV_CLEARNAV becomes true after the Filter_MPDU process decodes a packet containing a NAV and determines that the NAV needs to be set or cleared. After setting or clearing the NAV the process again returns to the state it left from.

There are two ways to go to another state from Cs_noNav. First, the condition RCVDSETNAV becomes true, indicating that the process now has a non-zero NAV. This will take the process to the Cs_Nav state and set the NAV timer. The second way to leave Cs_noNav is for the channel to physically be detected as idle. When this happens the process sets a timer based on the current value for the IFS, and then goes to the Wait_IFS state.

In the Wait_IFS state, there are three possibilities: (1) the condition RCVDSETNAV could become true, (2) the IFS timer could expire or be inactive, or (3) the channel could again become busy. If the RCVDSETNAV becomes true, the process goes to the Cs_Nav state (even though the channel is only virtually busy) and sets the NAV timer. If the IFS timer expires or it is not active, the process will declare the channel idle, start a slot timer (to begin counting slots) and go to the noCs_noNav state. If the channel is detected as busy prior to the IFS timer expiring, the process will go to the Cs_noNav state. When the IFS timer does expire in the Cs_noNav state, it will be ignored.

The Cs_Nav state will be left if the NAV timer expires or the channel is detected as idle. In the first case, the process will set the IFS timer and go to the Cs_noNav state. In the second case, the process will simply go to the noCs_Nav state.

The noCs_Nav state will be exited if the channel is detected as busy. In that case, the process will go to the Cs_Nav state. The other way to exit this state is for the NAV timer to expire. When this happens, the process will set the IFS timer and go to the state Wait_IFS.

The noCs_noNav state will be exited if the channel is detected as busy. When this happens, the process sends a busy interrupt to the process Data_Pump and goes to Cs_noNav. If the RCVDSETNAV condition becomes true in this state, the NAV timer is set with the appropriate value, a busy interrupt is sent, and the process goes to noCs_Nav. The SDL description of Channel_State are shown in Figures 16 and 17.

2.2.9 Tx_Coordination_sta

The Tx_Coordination_sta process model is shown in Figure 18. After Start, Tx_Coordination_sta goes to the TxC_Idle state. In this state, if VALID_INTERRUPT_ ALL_CACHE is true, the process will service any interrupts that have accumulated on the interrupt stack by going to the Process Interrupts state. Interrupts are only processed from the TxC_Idle state. In all other states, any interrupts not used by the state for transitions are saved for later servicing. The Cf_Response, Wait_Sifs, Wait_TxDone, and TxC_Cfp states are currently not used. These states would occur during the contention free period (CFP) of IEEE 802.11. The CFP is not implemented in the model at this time.

The process will leave TxC_Idle when it detects that the station queue has a packet to transmit (i.e., PDUREQUEST is true). It will request the packet be delivered from the queue and then enter the state GET_PACKET to wait for its arrival. From here, if NOELIGIBLEPACKET becomes true, the process returns to TxC_Idle. NOELIGIBLE PACKET could become true if, during the CFP (which is not implemented), a packet arrives for transmission but it is not of the correct class to be transmitted during the CFP. Once SENTPACKET is true, the process obtains the packet, makes a copy of it (for later retransmission if necessary), and then goes to the Wait_Mpdu_Backoff state. It stays in Wait_Mpdu_Backoff while the backoff timer is non-zero. When the backoff timer is determined to be zero (i.e., INTRPTNOTINBACKOFF is true), the packet is sent to the process Data_Pump for transmission and the Wait_Pdu_Sent state is entered. After

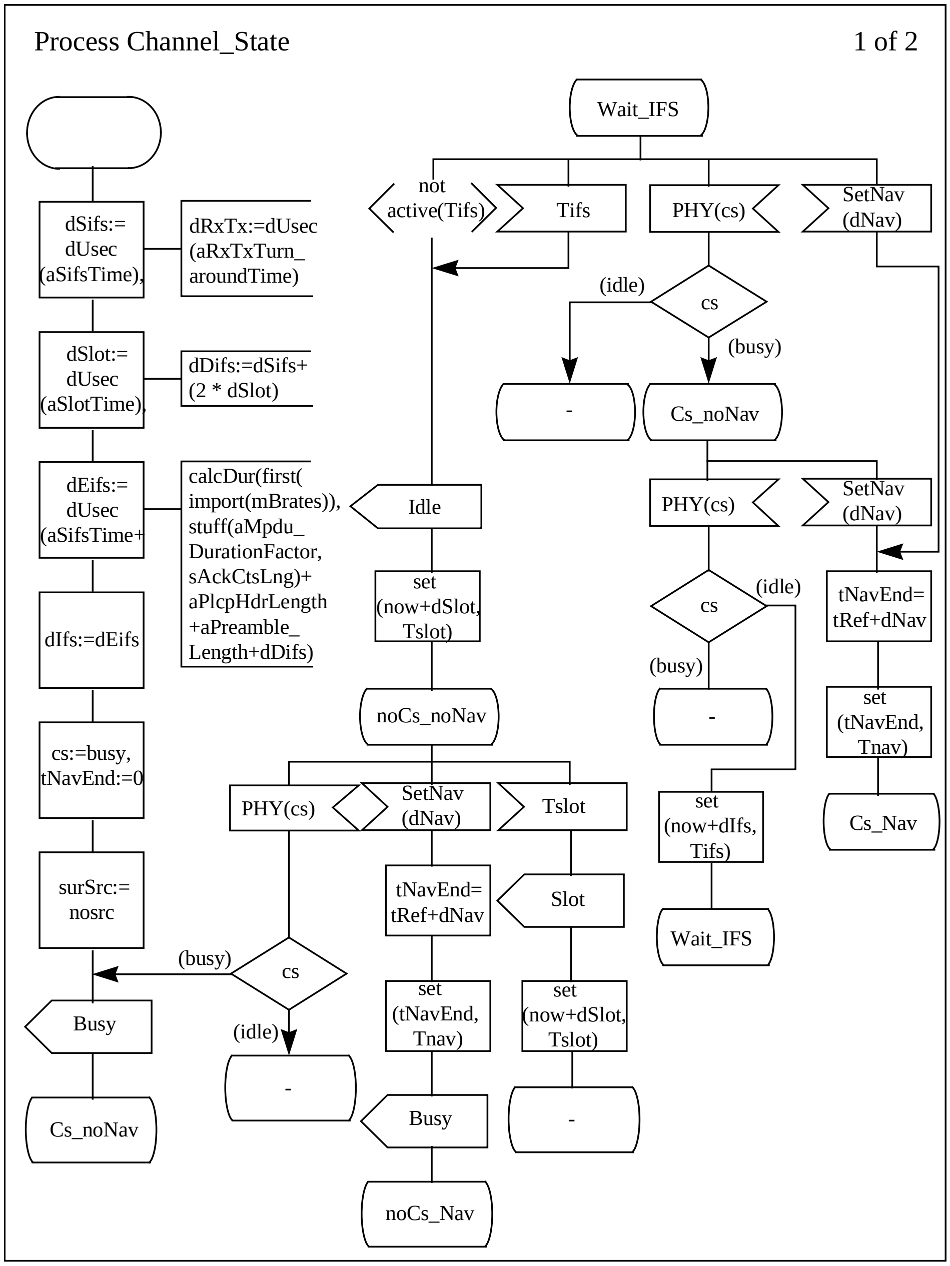

Figure 16: Channel_State - SDL Description (1 of 2)

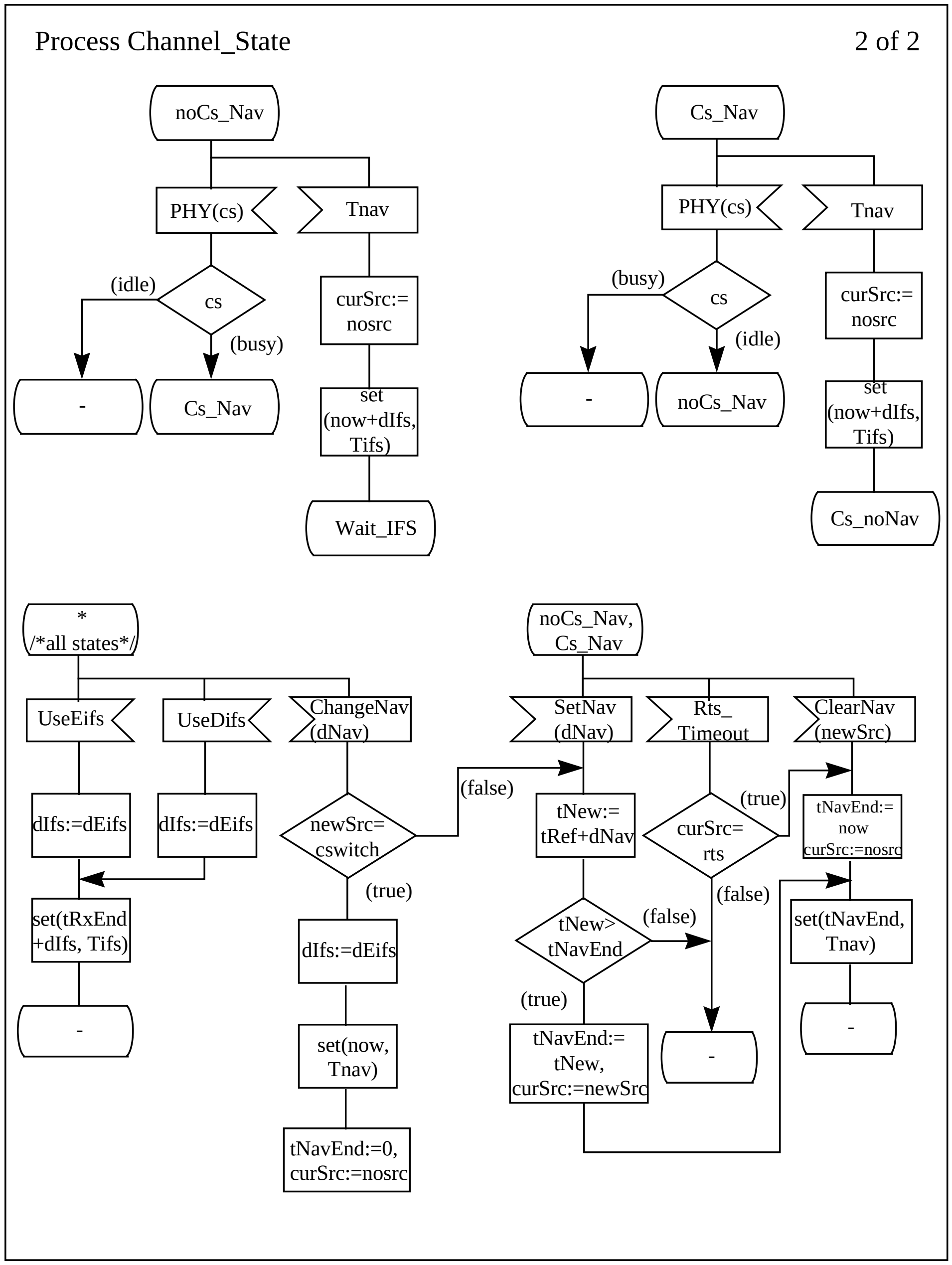

Figure 17: Channel_State - SDL Description (2 of 2)

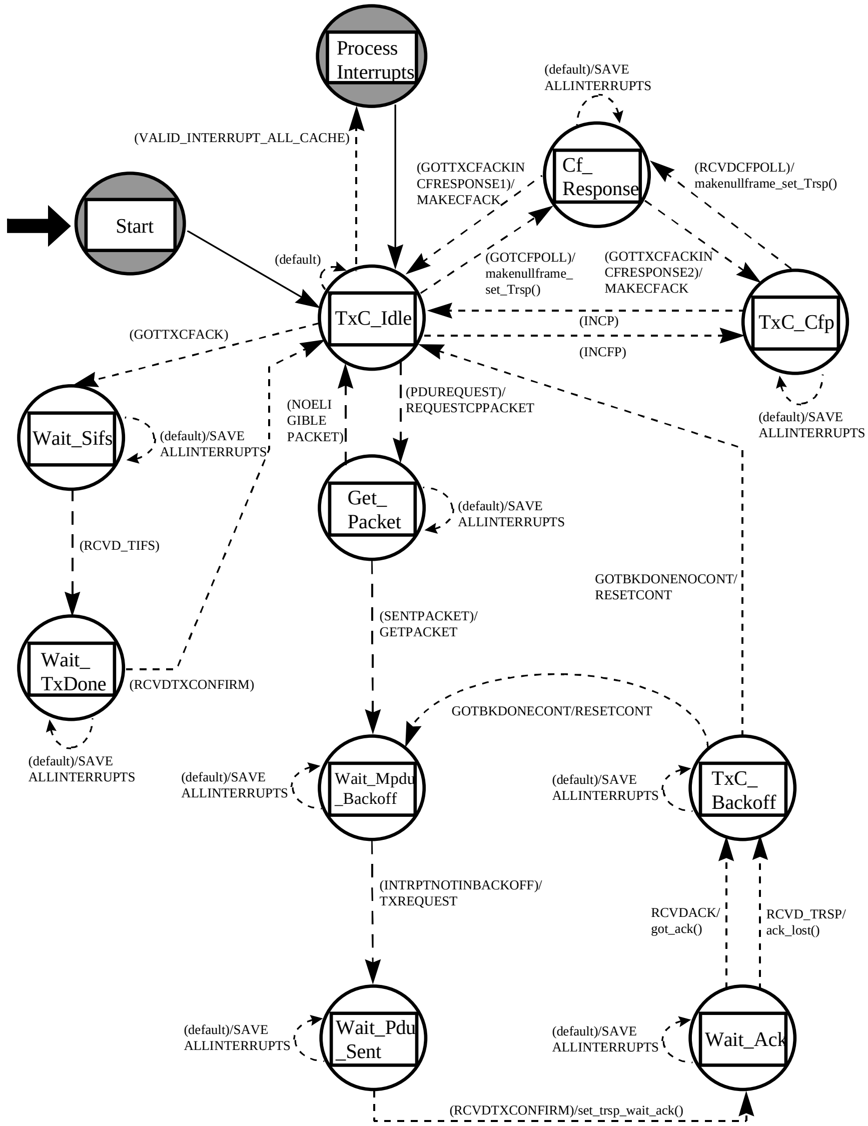

Figure 18: Tx_Coordination_sta Process Model

Data_Pump sends the packet, RCVDTXCONFIRM will become true. A timer is then set to wait for the ACK from the receiving station and the process goes to Wait_Ack. If the timer expires in this state, the packet is assumed to have been lost and a flag is set to reflect that assumption. If the ACK is received, the same flag is set to reflect a received ACK and the process goes to TxC_Backoff. In this state, another backoff count is obtained and the station waits for the backoff timer to expire. When this occurs, based on the value of the flag, the process goes to TxC_Idle, if an ACK was received, or to Wait_Mpdu_Backoff, if the ACK was not received.

The SDL description of Tx_Coordination_sta are shown in Figures 19 and 20.

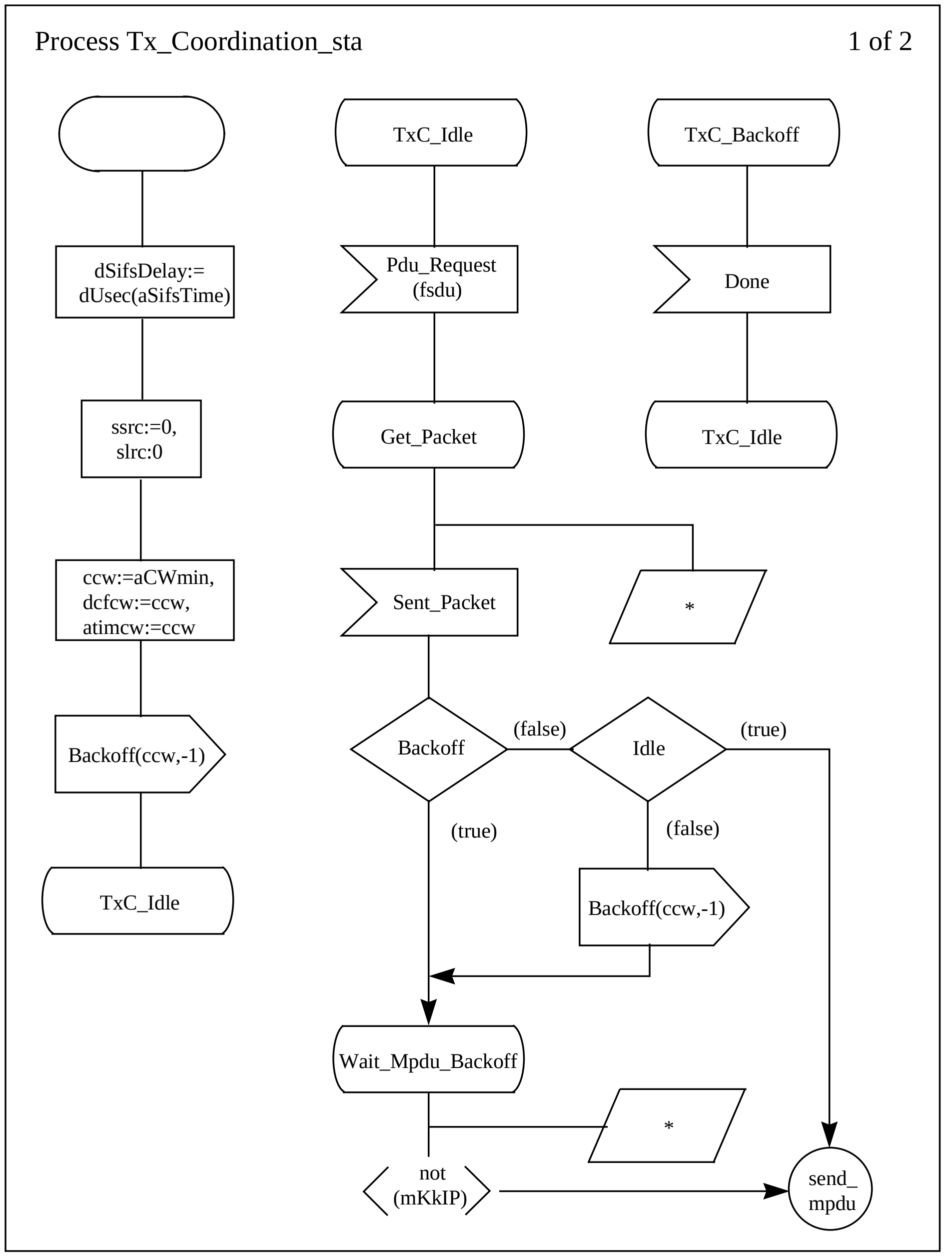

Figure 19: Tx_Coordination_sta - SDL Description (1 of 2)

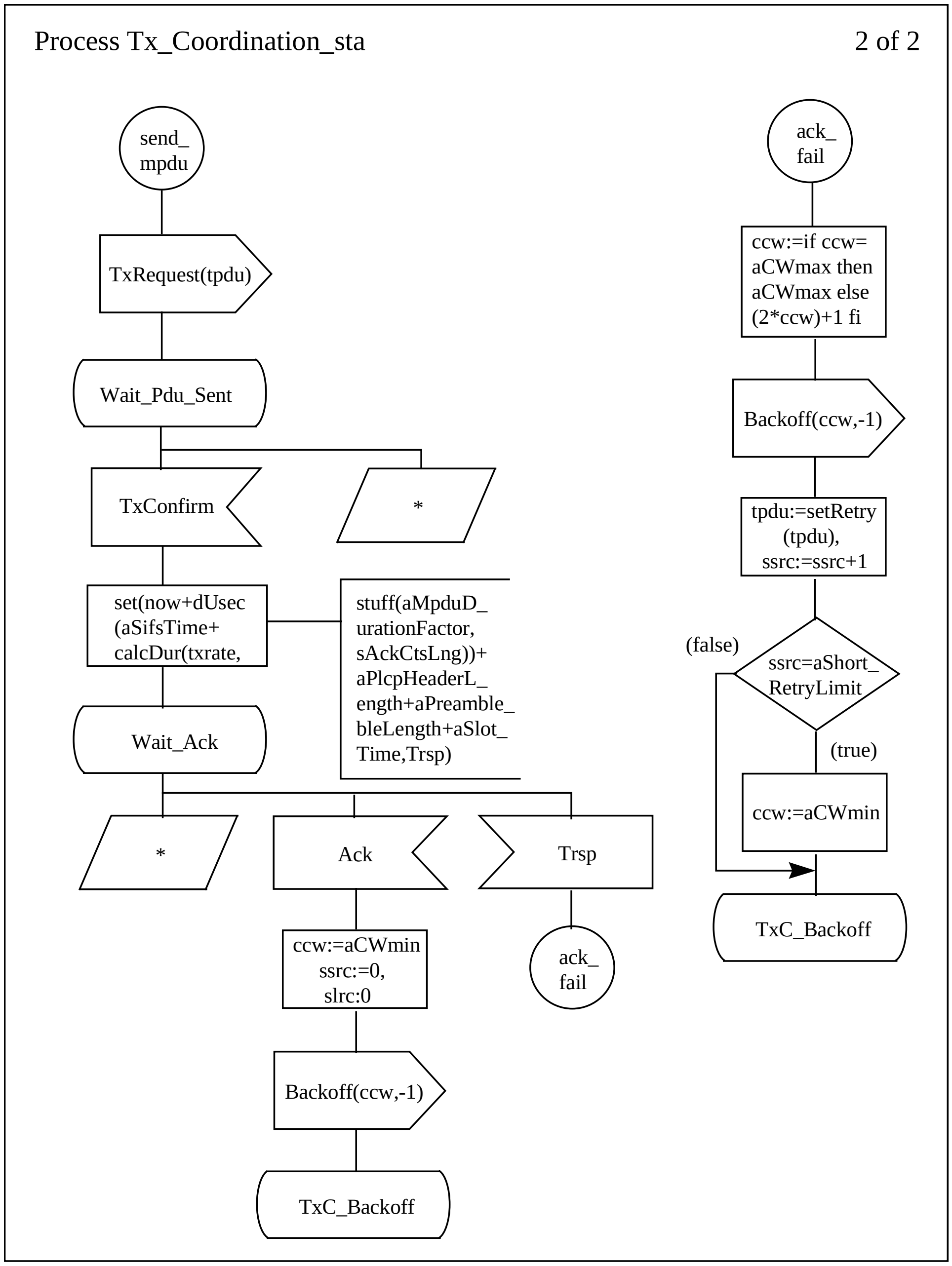

Figure 20: Tx_Coordination_sta - SDL Description (2 of 2)

2.2.10 Rx_Coordination

The process model for Rx_Coordination (Figure 21) begins in Start and goes to Rx_Idle. If an interrupt occurs or there are any saved interrupts (VALID_INTERRUPT_

ALL_CACHE) the process will transition to the Process_Interrupt state, process the interrupts, and return to RxC_Idle. The RCVDNEEDACKNOCACHE condition being

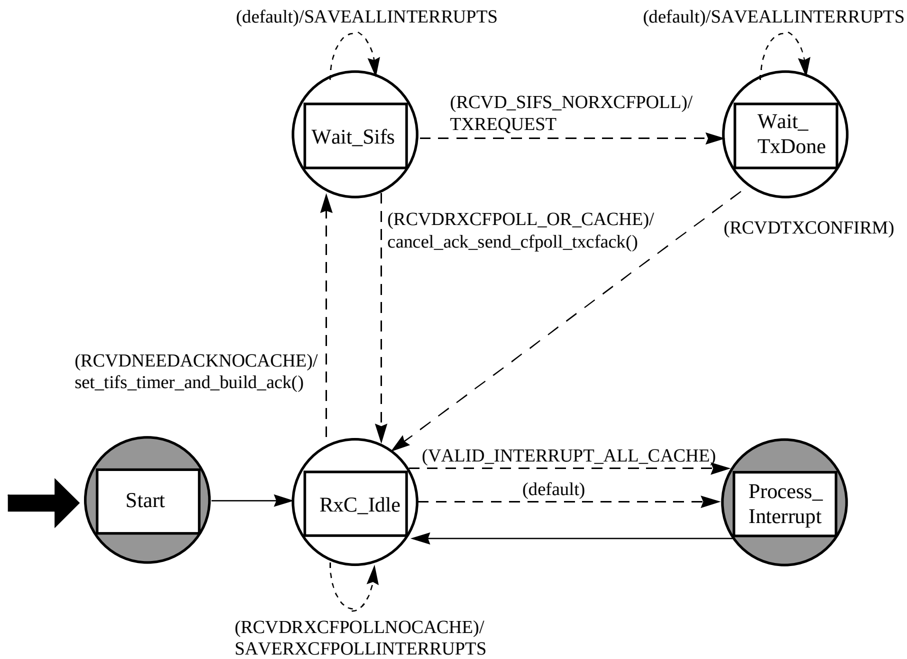

Figure 21: Rx_Coordination Process Model

true indicates that an ACK needs to be sent in response to a received packet. The process will set a timer, build the ACK and go to the Wait_Sifs state. The RCVDRXCFPOLLNOCACHE condition occurs in the (currently unimplemented) CFP. After the timer expires the process will send the ACK packet for transmission and enter the Wait_TxDone state. After receiving confirmation of packet transmission, the process leaves Wait_TxDone and returns to RxC_Idle. The SDL description of Rx_Coordination is shown below in Figure 22.

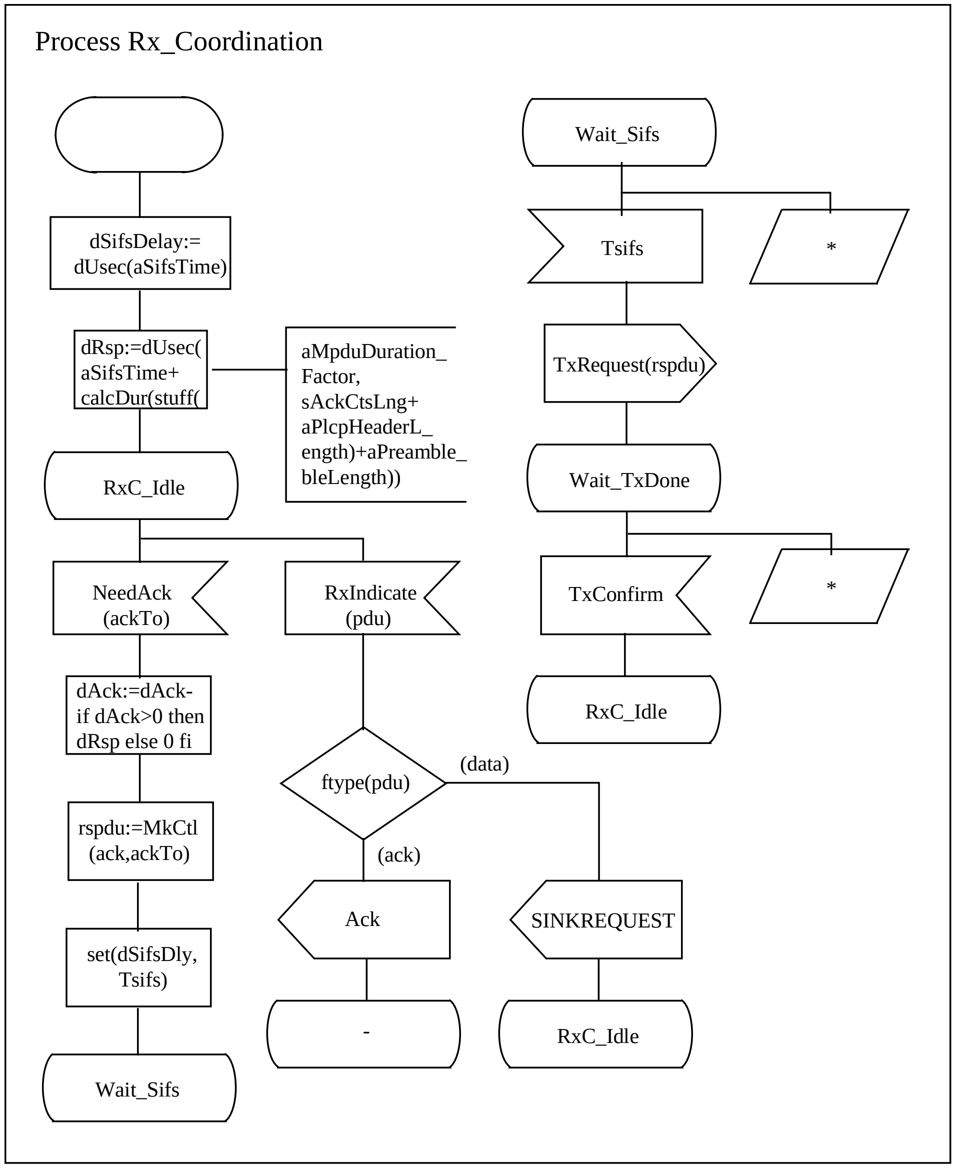

Figure 22: Rx_Coordination - SDL Description

2.2.11 Packet Format

The packets that are transmitted in the simulation model contain several fields, some of which are specified by IEEE 802.11 and some which are used to record information about the packet as it moves through the network. The packet format used in the simulation model is shown below in Table 1.

Table 1: Simulation Model Packet Format

Field Name	Data Type	Nomenclature
Type	integer	Packet type
Subtype	integer	Packet subtype
Address 1	integer	Address of the destination station
Address 2	integer	Address of the source station
Duration/ID	integer	The Network Allocation Vector
Deadline	double	Packet deadline
Retry	integer	Retry bit
Sequence Number	integer	Packet sequence number
Fragment Number	integer	Packet fragment number
Timestamp	double	General purpose timestamp
endRx	double	Time reception ended
Class	integer	Packet class
Xmit Duration	double	Transmit time
Debug Packet	integer	Debug packet flag
Data Octets	integer	Number of data octets in packet
Attempts	integer	Number of attempts to send original packet
Creation Time	double	Time original packet was created
Transmit Start Time	double	Starting time for current transmission
HOL Time	double	Time packet reached head-of-line in transmit queue
Deadline Variance	double	Allowable deadline variance

2.2.12 Traffic_Monitor

The purpose of Traffic_Monitor is to gather statistics about the network. It acts as a network "sniffer" or oracle by listening to all transmissions and recording traffic statistics. The OPNET node model for the Traffic_Monitor is show below in Figure 23.

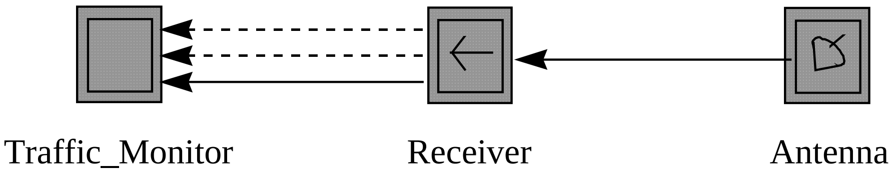

Figure 23: Traffic_Monitor Node Model

The Traffic_Monitor process model (Figure 24) begins in Start and then goes to the Idle state. It will stay there until it detects a corrupt packet (i.e., BADPACKET becomes true) upon which it will set a bad packet flag and return to Idle. Once it actually receives the packet, it goes to Process Packet, records all the statistics and then returns to Idle. The only statistic that cannot be determined from listening to the network traffic is the number of transmission attempts stations make. This is because several stations may

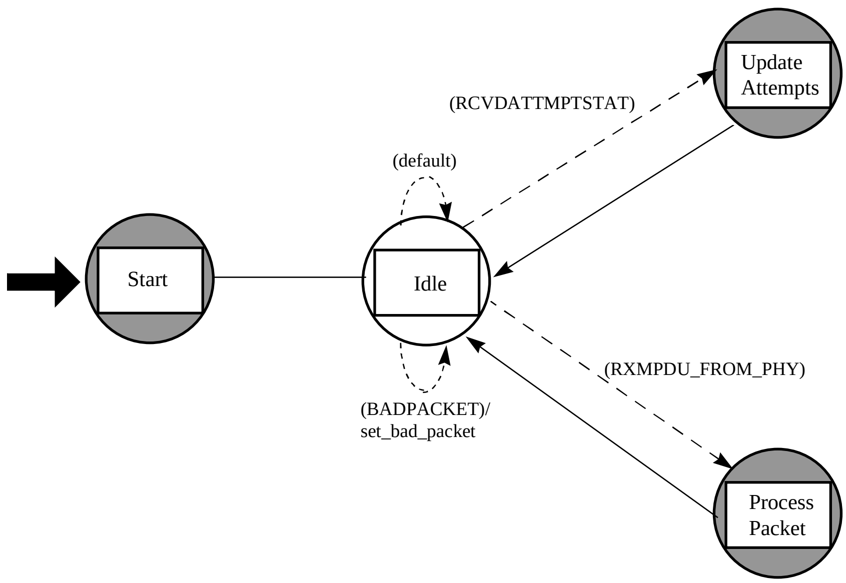

Figure 24: Traffic_Monitor Process Model

attempt transmission which results in a collision; by listening to the network traffic one can only determine that two or more stations attempted to transmit. After transmitting, each station sends a transmission attempt signal to this process to record the attempt (i.e., RCVDATTMPTSTAT). After recording the statistic, Traffic_Monitor returns to Idle.

3 Simulation Parameters

There are two types of simulation parameters. One type controls the simulation model such as simulation length and data collection start time. The other defines a characteristic of the network such as the number of stations or offered load. These parameters are discussed in this section. The parameters are organized by node and process models. The parameters are presented within the node or process model section in which they are originally defined.

3.1 IEEE80211STA

This is the node model which defines an IEEE 802.11 station within the simulation model. It defines the interconnections between the process models. It is shown in Figure 2. Over 20 different parameters can be varied, most of which are IEEE 802.11 parameters. These parameters are described in the following sections.

3.1.1 aCCATime

This parameter specifies the time within every slot the clear channel assessment (CCA) mechanism has to determine whether the channel is idle. Its default value is 15 usec.

3.1.2 aCWmax

This is the maximum value (in slots) of the contention window. Its default value is 1023.

3.1.3 aCWmin

This is the minimum value (in slots) of the contention window. Its default value is 31.

3.1.4 aMPDUMaxLength

This specifies the maximum supported MAC protocol data unit (PDU) length. Its default value is 8191 octets.

3.1.5 aPLCPHeaderLength

This parameter specifies the length of the header in the physical layer convergence protocol (PLCP). Its default value is 48 bits.

3.1.6 aPreambleLength

This parameter specifies the length in bits of the physical layer frame preamble. Its default value is 144 bits.

3.1.7 aRxPLCPDelay

This parameter specifies the delay introduced by the PLCP during frame reception. Its default value is 1 usec.

3.1.8 aRxRFDelay

This parameter determines the RF delay introduced during a reception. Its default value is 5 usec.

3.1.9 aRxTxTurnaroundTime

This parameter specifies the time the radio requires to change modes from receive to transmit or transmit to receive. The default value is 20 usec.

3.1.10 aShortRetryLimit

This parameter specifies how many attempts will be made to transmit a short packet. Its default value is 7.

3.1.11 aSifsTime

This parameter defines the short inter-frame space (SIFS) time. Its default value is 10 usec.

3.1.12 aSlotTime

This parameter specifies the width of a slot. Its default value is 20 usec.

3.1.13 aTxPLCPDelay

This parameter specifies the delay introduced by the PLCP. Its default value is 5 usec.

3.1.14 aTxRFDelay

This parameter determines the RF delay introduced during transmissions. Its default value is 1 usec.

3.1.15 ACK_Length

This determines the length, in bits, of an ACK packet. The value specified here will override the value specified in sAckCtsLng. Its default value is Default which uses the sAckCtsLng value.

3.1.16 ACK_Timeout

This determines the length of time the station will wait for an ACK. This is normally calculated by the station but can be specified. Its default value is Default, which uses the internal calculation.

3.1.17 Backoff Algorithm

This parameter changes the algorithm used to determine contention window values. Its default value is Standard IEEE 802.11.

3.1.18 MACHeaderLength

This parameter determines the length, in bits, of a MAC packet header. Normally the value is calculated internally. Specifying a value here will override this internal calculation. Its default value is Default.

3.1.19 Number of Stations

This parameter informs the station how many stations are in the network. It is used to determine the set of valid station addresses to transmit to. Its default value is 0 and should be changed to reflect the number of stations in the network.

3.1.20 Packet format

This parameter determines the packet format to be used in the station. Its default value is IEEE80211frame. Note the simulation accesses many fields within this frame format. Specifying another frame format that does not include the fields in IEEE80211 will cause the simulation to fail.

3.1.21 Print Errored Packets

This Boolean parameter determines whether the station will print information about packets that have errors in them. Normally, packets with errors are discarded without warning. This can generate a large amount of output and should be used for debugging purposes only. Its default value is false.

3.1.22 Print Packets

This Boolean parameter determines whether the station will print information about packets transmitted and received by the station. This can generate a large amount of output and should only be used for debugging purposes. Its default value is false.

3.1.23 RxTxRate

This parameter specifies the receive and transmit rate for the station. It is integer valued and the RxTxRate multiplied by 500 kbit/sec determines the station's transmit and receive rate. Its default value is 2.

3.1.24 sAckCtsLng

This parameter specifies the length of an ACK and a clear-to-send CTS frame. Its default value is 112 bits.

3.1.25 sMaxMsduLng

This parameter specifies the maximum length, in octets, of the MAC service data unit (SDU). Its default value is 2304.

3.1.26 SA

This parameter specifies the station address (SA). A stations address is a positive integer and must be specified. If this parameter is left to the default value of -1, a simulation error will occur.

3.1.27 Sequence Number

This parameter determines the initial value for packet sequence numbers. Its default value is -1.

3.2 IEEE80211MON

This node model is the Traffic_Monitor discussed in Section A.2.2.11. It has two parameters that can be specified: Data Collection Start and Data Collection Stop. The default values are 5.0 and 1,000,000.0 seconds respectively.

3.3 Source

Within the source process model, the characteristics of the packets submitted for transmission are defined. There are a large number of parameters that can be specified, but they are largely redundant. The type of the parameter is enclosed in parenthesis. Sometimes the parameter names will include brackets (i.e., []). The brackets denote other parameters of the same type and purpose but with different names. For instance, ABC[1-3] (integer) would denote three parameters of type integer: ABC1, ABC2, and ABC3.

3.3.1 Active Source (Boolean)

The Source process model can be prevented from sending any data onto the network by setting this parameter to false. If true, Source will generate packets according to the values specified in the other parameters. The default value for this parameter is true.

3.3.2 [Hard Real Time, Soft Real Time, Data] Sources (integer)

This parameter specifies the number of independent sources for the three classes of data (i.e., hard, soft, and data). The number of sources may be from zero to three. The default value is zero.

3.3.3 [HRT[1-3], SRT[1-3], D[1-3]] Interarrival Distribution (string)

This parameter specifies the interarrival distribution for the packet sources. If there are 2 Data Sources specified (c.f., Section A.3.3.2), the D1 Interarrival Distribution and the D2 Interarrival Distribution will be used to generate packets. The distributions that can be specified are: Bernoulli, chi_square, constant, erlang, exponential, normal, Poisson, uniform, and uniform_int. The default value is exponential.

3.3.4 [HRT[1-3], SRT[1-3], D[1-3]] Interarrival Parameter [1,2] (double)

This parameter is used to specify the parameters for the interarrival distribution chosen in Section A.3.3.3 above. Up to two parameters can be specified. If a distribution only requires one parameter [HRT[1-3], SRT[1-3], D[1-3]] Interarrival Parameter 1 will be used. The default values for these parameters are zero.

3.3.5 [HRT[1-3], SRT[1-3]] Deadline Distribution (string)

This parameter specifies the deadline distribution for the packet sources. If there are 2 Hard Real Time Sources specified (c.f., Section A.3.3.2), the HRT1 Deadline Distribution and the HRT2 Deadline Distribution will be used to generate deadlines. The distributions that can be specified are: Bernoulli, chi_square, constant, erlang, exponential, normal, Poisson, uniform, and uniform_int. The default value is exponential.

3.3.6 [HRT[1-3], SRT[1-3]] Deadline Parameter [1,2] (double)

This parameter is used to specify the parameters for the deadline distribution chosen in Section A.3.3.5 above. Up to two parameters can be specified. If a distribution only requires one parameter [HRT[1-3], SRT[1-3]] Deadline Parameter 1 will be used. The default values for these parameters is zero.

3.3.7 [HRT[1-3], SRT[1-3], D[1-3]] Packet Size Distribution (string)

This parameter specifies the packet size distribution for the packet sources. If there are 2 Data Sources specified (c.f., Section A.3.3.2), the D1 Packet Size Distribution and the D2 Packet Size Distribution will be used to generate packets sizes. The distributions that can be specified are: constant, geometric, and uniform_int. The default value is constant.

3.3.8 [HRT[1-3], SRT[1-3], D[1-3]] Packet Size Parameter [1,2] (double)

This parameter is used to specify the parameters for the packet size distribution chosen in Section A.3.3.7 above. Up to two parameters can be specified. If a distribution only requires one parameter [HRT[1-3], SRT[1-3], D[1-3]] Packet Size Parameter 1 will be used. The default value for Parameter 1 is 200. The default value for Parameter 2 is zero.

3.4 Receiver, Transmitter, Antenna

The parameters for these OPNET object generally do not need to be changed. If they do need to be changed, the OPNET Node Editor is used. Within the simulation model, the transmitter and receiver pipeline has been left in the default configuration as supplied by MIL3 (except for one case noted below). The source files and routine names have, however, been changed to be specific to this simulation model. This allows the pipeline to be changed without affecting other models. Obviously, this means that the sources files will need to compiled prior to use.

3.4.1 Receiver

The default values for the Receiver parameters are as follows:

1) modulation - bpsk
2) channel.channel - channel[0]
3) channel.data rate - 1,000,000
4) channel.packet formats - all formatted, unformatted
5) channel.bandwidth - 22,000
6) channel.min frequency - 2,400
7) channel.spreading code - 1.0
8) channel.processing gain - channel bw/dr
9) noise figure - 1.0
10) ecc threshold - 1.0
11) ragain model - IEEE80211_ragain
12) power model - IEEE80211_power
13) bkgnoise model - IEEE80211_bkgnoise
14) inoise model - IEEE80211_inoise
15) snr model - IEEE80211_snr
16) ber model - IEEE80211_ber

The ber model has been modified so that no bit errors are introduced.

17) error model - IEEE80211_error

18) ecc model - IEEE80211_ecc

3.4.2 Transmitter

The default values for the Transmitter parameters are as follows:

1) modulation - bpsk
2) channel.channel - channel[0]
3) channel.data rate - 1,000,000
4) channel.packet formats - all formatted, unformatted
5) channel.bandwidth - 22,000
6) channel.min frequency - 2,400
7) channel.spreading code - 1.0
8) channel.power - 1.0
9) rxgroup model - IEEE80211_rxgroup
10) txdel model - IEEE80211_txdel
11) closure model - IEEE80211_closure
12) chanmatch model - IEEE80211_chanmatch
13) tagain model - IEEE80211_tagain
14) propdel model - IEEE80211_propdel

3.4.3 Antenna

The only value changed in the Antenna parameters is pattern. The value is IEEE80211 which implements a standard omnidirectional pattern.

4 Model Validation

The model was validated by comparing the performance metrics obtained by the model against those obtained in [BiF96]. Because [BiF96] was based on an earlier draft, several of that paper's parameters did not match those in the latest IEEE 802.11 standard. For the validation, the values of these parameters in our model were changed to match

[BiF96]. Figure 25 shows throughput for 5, 10 and 20 station networks. Figure 26 shows the average transmission attempts per packet for various values of the contention window, CWmin, and CWmax. Figure 27 shows saturation throughput versus number of stations for various values of the contention window, CWmin, and CWmax. The agree-

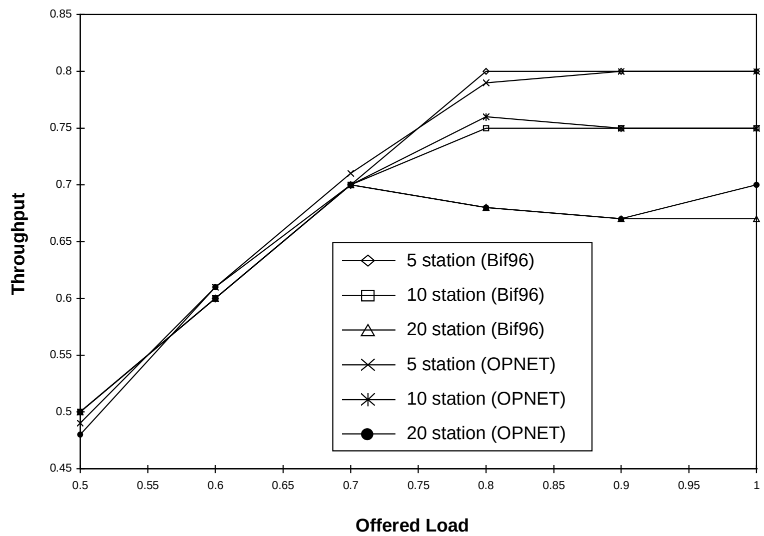

Figure 25: Throughput versus Offered Load

ment between the two models is quite good. In fact, the data obtained in the two uppermost plots in Figure 26 were so close to [BiF96] that our data and theirs overlap. Other performance metrics were compared as well and the agreement was similar.

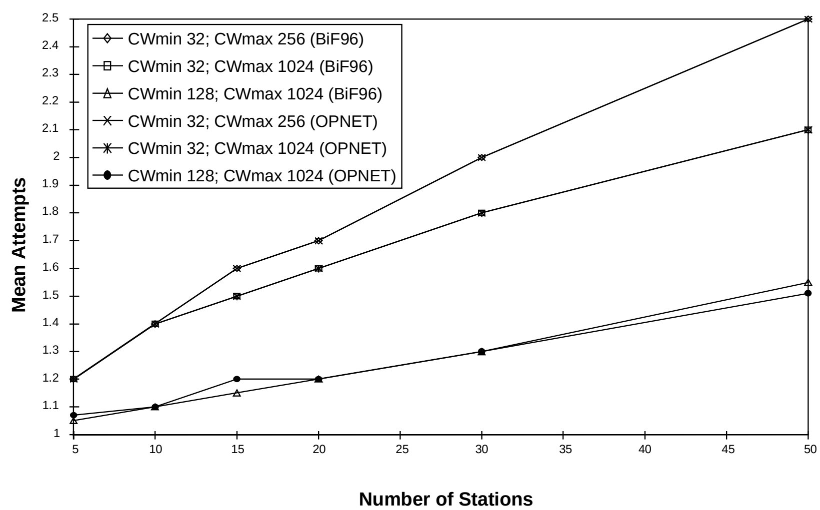

Figure 26: Mean Attempts per Packet

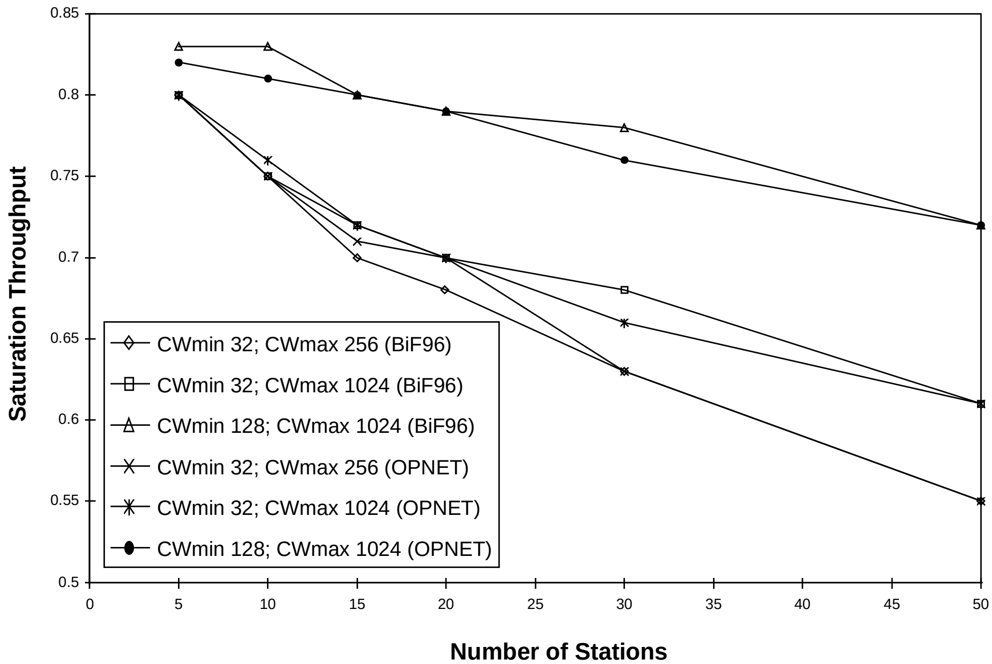

Figure 27: Saturation Throughput

5 Summary

This document describes the IEEE 802.11 simulation model. It begins with a discussion of the simulation tool, OPNET, in Section 1. In Section 2 the simulation model is described. The node and process models are presented and the model behavior is described using SDL. Section 3 discusses the simulation parameters which may be varied within the model. Validation of the model is discussed in Section 4.

References

BiF96 G. Bianchi, L. Fratta, M. Oliveri, “Performance Evaluation and Enhancement of the CSMA/CA MAC Protocol for 802.11 Wireless LANs,” *The Seventh IEEE International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC ’96*, pp. 392-396, Oct. 1996.

EHS97 J. Ellsberger, D. Hogrefe, A. Sarma, *SDL, Formal Object-Oriented Language for Communicating Systems*, Prentice Hall Europe, Hertfordshire, UK, 1997.

IEE97 Editors of IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Draft Standard 802.11, P802.11/D6.1*, Institute of Electrical and Electronics Engineers, Inc., New York, May 9, 1997.

MIL97 OPNET Modeler, *MIL 3, Inc*., 3400 International Drive NW, Washington D. C., 20008.