Auto Rate Fallback Algorithm for the IEEE 802.11a Standard

A.J. van der Vegt
High Performance Computing Group
Faculty of Physics and Astronomy
Utrecht University

Supervisors:
dr ir C.T.A.M. de Laat
ir A. Kamerman

Abstract

The IEEE (Institute for Electrical and Electronics Engineers) 802.11a wireless local area networking (WLAN) standard describes a wireless network that can support the simultaneous use of high data rate applications in a mobile, multi-user environment. The 802.11a-compliant WLANs will be able to support data rates ranging from 6 to 54 Mbps. The IEEE 802.11a WLAN standard does not specify an Auto Rate Fallback (ARF) algorithm.

This report describes an evaluation of ARF for 802.11a WLANs. A software model of the 802.11a WLAN standard developed for the OPNET simulation tool has been used. The model includes both the Medium Access Control (MAC) and Physical (PHY) layers of the 802.11a standard. Three different ARF algorithms based on the number of missed acknowledgements are proposed. The performance of the ARF algorithm is analysed in two different scenarios.

The results of these simulations show the need for a so called "Probation State". An ARF algorithm with easy fallback and hard fall forward parameters will give the best throughput results.

Contents

List of Abbreviations

ACK ACKnowledgment

AP Access Point

ARF Auto Rate Fallback

ARP Address Resolution Protocol

BER Bit Error Rate

BPSK Binary Phase Shift Keying

BSA Basic Service Area

BSS Basic Service Set

CPU Central Processor Unit

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance

CTS Clear-To-Send

CW Contention Window

DCF Distributed Coordination Function

DIFS DCF IFS

DR Default Rate

DS Distribution System

ECC Error Correction Code

EMA External Model Access

ESS Extended Service Set

FFT Fast Fourier Transform

FR Fallback Rate

FTP File Transfer Protocol

HIPERLAN HIgh PErformance Radio LAN

IBSS Independent BSS

ICI Interface Control Information

IEEE Institute of Electrical and Electronics Engineers

IFFT Inverse Fast Fourier Transform

IFS Inter Frame Space

IP Internet Protocol

LAN Local Area Network

LLC Logical Link Control

MAC Medium Access Control

MMAC Multimedia Mobile Access Communications

MSDU MAC Service Data Units

MT Mobile Terminal

NAV Network Allocation Vector

OFDM Orthogonal Frequency Division Multiplexing

PCF Point Coordination Function

PDF Probability Density Function

PHY PHYsical layer

PIFS PCF IFS

QAM Quadrature Amplitude Modulation

QPSK Quadrature Phase Shift Keying

RTS Request-To-Send

SIFS Short IFS

SNR Signal-to-Noise Ratio

STD State Transition Diagram

TCP Transmission Control Protocol

UDP User Datagram Protocol

WLAN Wireless LAN

Chapter 1

Introduction

To support the consumer demand for high data rate communications, new Wireless Local Area Network (WLAN) standards have emerged. The most well developed standard is the IEEE 802.11 WLAN standard [1]. The 802.11 standard focuses on the Medium Access Control (MAC) and PHY (PHYsical layer) for Access Point (AP) based networks and ad-hoc networks. The original version of this standard provides data rates of 1 and 2 Mb/s. The 802.11b extension [2] of the standard supports data rates of 1, 2, 5.5 and 11 Mb/s for operation in the 2.4 GHz band, and the 802.11a extension [3] supports data rates of 6, 9, 12, 18, 24, 36, 48, and 54 Mb/s for operation in the 5 GHz band. The 802.11g task group has started with a new extension that will support the same data rates as with 802.11a and 802.11b for operation in the 2.4 GHz band. Other high data rate WLAN standards are the European HIPERLAN/2 [4] and the Japanese Multimedia Mobile Access Communications (MMAC) protocol.

Different data rates have their own benefit in wireless communications. Communicating at a higher data rate cannot be said to be superior to communicating at a lower data rate. This is because lower data rates allow communication in worse channel conditions and at larger coverage ranges. Thus, an Auto Rate Fallback (ARF) algorithm which can determine the optimum data rate for a given channel condition, will improve the overall performance. The 802.11a standard does not specify such an ARF algorithm. This report will discuss the simulation of an ARF algorithm.

Networking technology has become too complex for traditional analytical methods of "rules of thumb" to yield an accurate understanding of system behaviour, therefore we chose for simulation. The only simulator that includes a model of the IEEE 802.11a protocol is OPNET Modeler. This model for OPNET was developed at the Philips Research Labs [5], and further refined by Bryan Braswell [6]. The 802.11a model was used to test the ARF algorithm based on missed acknowledgments (ACKs). The ARF algorithm has been added to this existing model.

Chapter 2 gives an outline of the relevant parameters of the IEEE 802.11a protocol. Chapter 3 gives a detailed overview of the OPNET Modeler environment. Results of the simulations are presented in Chapter 4. The final chapter gives the conclusions drawn from the simulations.

Four appendices have been included. The first Appendix gives the ARF Tables, which determine the behaviour of the ARF algorithm. The second, third and fourth Appendix list some of the most relevant code with respect to the added ARF algorithm.

Chapter 2

IEEE 802.11a Wireless LANs

The IEEE 802.11 WLAN standard [1] focuses on the MAC and PHY for Access Point (AP) based networks and ad-hoc networks.

Extension 802.11a [3] is for a high data rate OFDM (Orthogonal Frequency Division Multiplexing modulation) PHY standard providing data rates in the range of 6 to 54 Mb/s in the 5 GHz band. The 802.11a PHY, and the other 802.11 PHYs have the same MAC layer using Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA).

This chapter will concentrate on the, in our context, essential elements of the 802.11a standard as also described in [7].

2.1 IEEE 802.11 Topology

The Basic Service Set (BSS) is the fundamental building block of the IEEE 802.11 architecture. A BSS is defined as a group of stations that are under the direct control of a single coordination function (i.e., a Distributed Coordination Function, DCF) which is defined below. The geographical area covered by the BSS is known as the Basic Service Area (BSA), which is analogous to a cell in a cellular communications network. Conceptually, all stations in a BSS can communicate directly with all other stations in a BSS. However, transmission medium degradations due to multipath fading, or interference from nearby BSSs reusing the same physical layer characteristics, can cause some stations to appear hidden from other stations.

The 802.11 standard supports the following two topologies:

- Independent Basic Service Set (IBSS) networks
- Extended Service Set (ESS) networks

IBSS is the formal name of an ad hoc network in the IEEE 802.11 standard. An ad hoc network is a deliberate grouping of stations into a single BSS for the purpose of internetworked communications without the aid of an infrastructure network.

We will concentrate on the ESS as shown in Figure 2.1. In contrast to the ad hoc network, infrastructure networks are established to provide wireless users with specific services. Infrastructure networks in the context of IEEE 802.11 are established using APs. The AP is analogous to the base station in a cellular communications network. An AP supports range extension by providing the integration points necessary for network connectivity between multiple BSSs, thus forming an ESS. The ESS has the appearance of

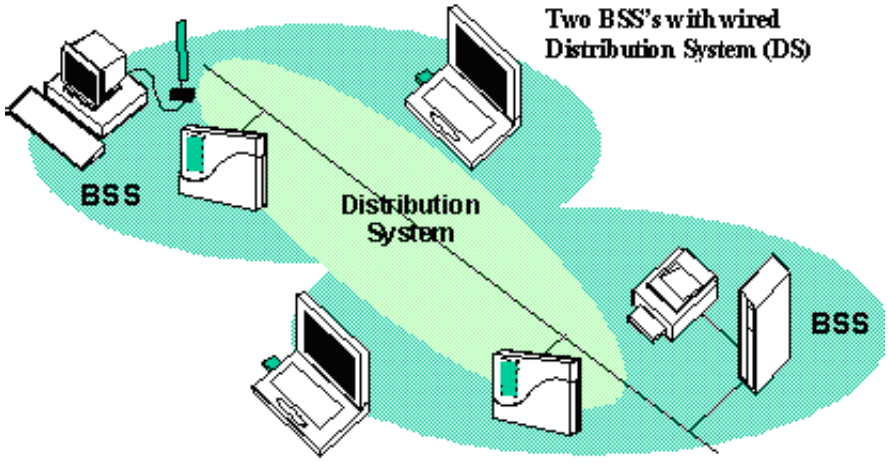

Figure 2.1: ESS with wired distribution system.

one large BSS to the Logical Link Control (LLC) sub layer of each station. The ESS consists of multiple BSSs that are integrated together using a common Distribution System (DS). The DS can be thought of as a backbone network that is responsible for MAC-level transport of MAC Service Data Units (MSDUs). The DS, as specified by IEEE 802.11, is implementation-independent. An ESS can also provide gateway access for wireless users into a wired network such as the Internet.

2.2 Medium Access Control Layer

The 802.11 basic medium access behaviour allows interoperability between compatible PHYs through the use of the CSMA/CA protocol and a random back-off time following a busy medium condition. In addition, all directed traffic uses immediate positive acknowledgment (ACK frame), where the sender schedules a retransmission if no ACK is received. The 802.11 CSMA/CA protocol is designed to reduce the collision probability between multiple stations accessing the medium at the point in time where collisions would most likely occur. Collisions are most likely to happen just after the medium becomes free, just after busy medium conditions. This is because multiple stations would have been waiting for the medium to become available again. Therefore, a random back-off arrangement is used to resolve medium contention conflicts. The 802.11 MAC defines special functional behaviour for fragmentation of packets, medium reservation via RTS/CTS (request-to-send/clear-to-send) polling interaction and point coordination (for time-bounded services).

In this section, we will discuss the different relevant functions of MAC layer.

2.2.1 Inter Frame Spacing

In Figure 2.2 inter frame spacing (IFS) required for MAC protocol are given. Their lengths are, for IEEE 802.11a:

- SIFS = Short Inter Frame Space = 16 μs
- PIFS = PCF (Point Coordination Function) Inter Frame Space
  = SIFS + Slot Time (25 μs)

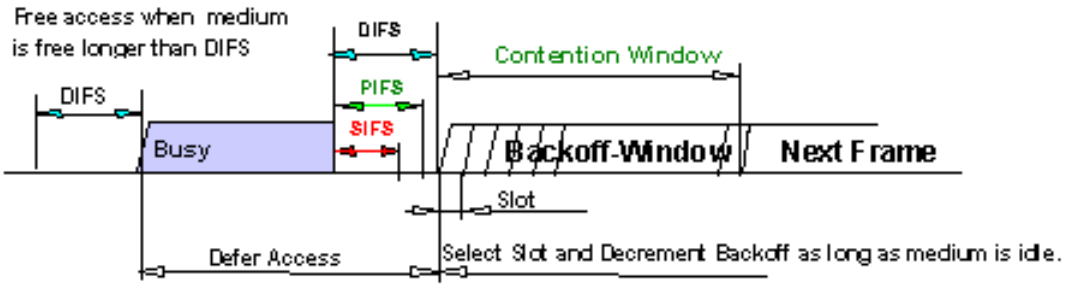

Figure 2.2: IEEE 802.11 Inter Frame Space.

- DIFS = DCF (Distributed Coordination Function) Inter Frame Space
  = PIFS + Slot Time (34 μs)

The back-off timer is expressed in terms of number of time slots (9 μs). We will concentrate on the DCF, as this is the most common Coordination Function used nowadays, and, therefore, used during our simulations. For further information on the PCF see [7].

2.2.2 Distributed Coordination Function

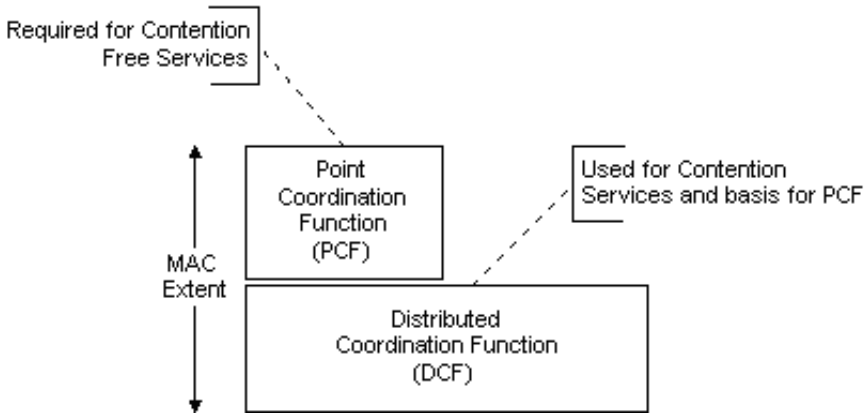

Figure 2.3: MAC Architecture.

The DCF is the fundamental access method used to support asynchronous data transfer on a best effort basis. As identified in the specification, all stations must support the DCF. The MAC architecture is depicted in Figure 2.3, where it is shown that the DCF is positioned directly on top of the physical layer and supports contention services. Contention services imply that each station with a MSDU queued for transmission must contend for access to the channel and, once the MSDU is transmitted, must recontend for access to the channel for all subsequent frames. Contention services promote fair access to the channel for all stations.

Priority access to the wireless medium is controlled through the use of IFS time intervals between the transmission of frames. The IFS intervals are mandatory periods of idle time on the transmission medium. Three IFS intervals, as explained in Section 2.2.1, are specified in the standard: SIFS, PIFS, and DIFS. The SIFS interval is the smallest IFS, followed by

PIFS and DIFS, respectively. Stations only required to wait a SIFS have priority access over those stations required to wait a PIFS or DIFS before transmitting; therefore, SIFS has the highest-priority access to the communications medium. For the basic access method, when a station senses the channel is idle, the station waits for a DIFS period and samples the channel again. If the channel is still idle, the station transmits an MPDU. The receiving station calculates the checksum and determines whether the packet was received correctly. Upon receipt of a correct packet, the receiving station waits a SIFS interval and transmits a positive ACK back to the source station, indicating that the transmission was successful. Figure 2.4 is a timing diagram illustrating the successful transmission of a data frame. When the data frame is transmitted, the duration field of the frame is used to let all stations in the BSS know how long the medium will be busy. All stations hearing the data frame adjust their Network Allocation Vector (NAV) based on the duration field value, which includes the SIFS interval and the ACK following the data frame.

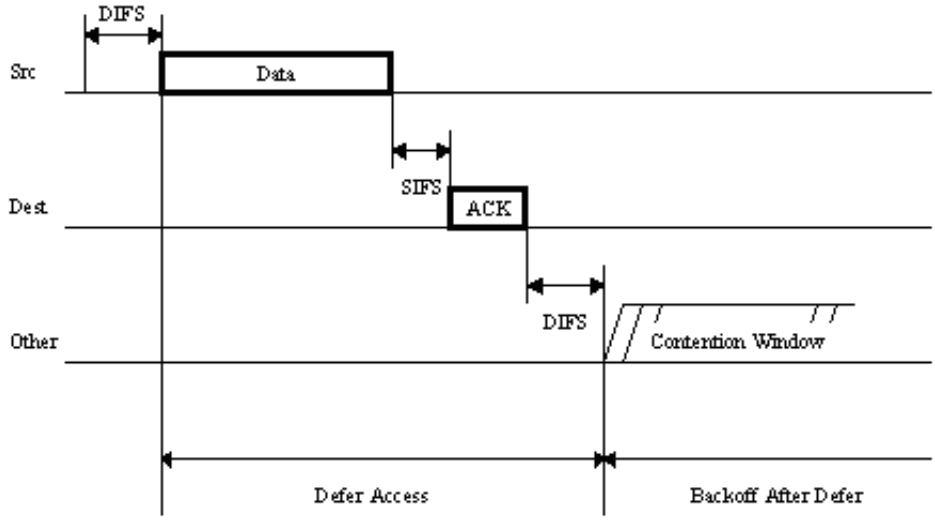

Figure 2.4: Transmission of a MPDU.

The collision avoidance portion of CSMA/CA is performed through a random back off procedure. If a station with a frame to transmit initially senses the channel to be busy; then the station waits until the channel becomes idle for a DIFS period, and then computes a random back off time. For IEEE 802.11, time is slotted in time periods that correspond to a Slot-Time. The Slot-Time used in IEEE 802.11 is much smaller than an MPDU and is used to define the IFS intervals and determine the back off time for stations. The Slot-Time is different for each physical layer implementation. The random back off time is an integer value that corresponds to a number of time slots. Initially, the station computes a back off time in the range 0–15 for 802.11a. After the medium becomes idle after a DIFS period, stations decrement their back off timer until the medium becomes busy again or the timer reaches zero. If the timer has not reached zero and the medium becomes busy, the station freezes its timer. When the timer is finally decremented to zero, the station transmits its frame. If two or more stations decrement to zero at the same time, a collision will occur which lead to missing ACKs, and each station will have to generate a new back off time in the range 0–31 for 802.11a times the Slot Time period. The generated back off

time corresponds to a uniform distributed integer multiple of Slot Time periods. For the next retransmission attempt, the back off time grows to 0–63 Slot Time periods and so on with at maximum the range 0–1023, however retry counter limits will prevent this range to be used in practice. The idle period after a DIFS period is referred to as the contention window (CW). The advantage of this channel access method is that it promotes fairness among stations, but its weakness is that it doesn't support time bound services. Fairness is maintained because each station must recontend for the channel after every transmission of an MSDU. All stations have equal probability of gaining access to the channel after each DIFS interval. Time-bounded services typically support applications such as packetized voice or video that must be maintained with a specified minimum delay. With DCF, there is no mechanism to guarantee minimum delay to stations supporting time-bounded services.

2.3 IEEE 802.11a Physical Layer

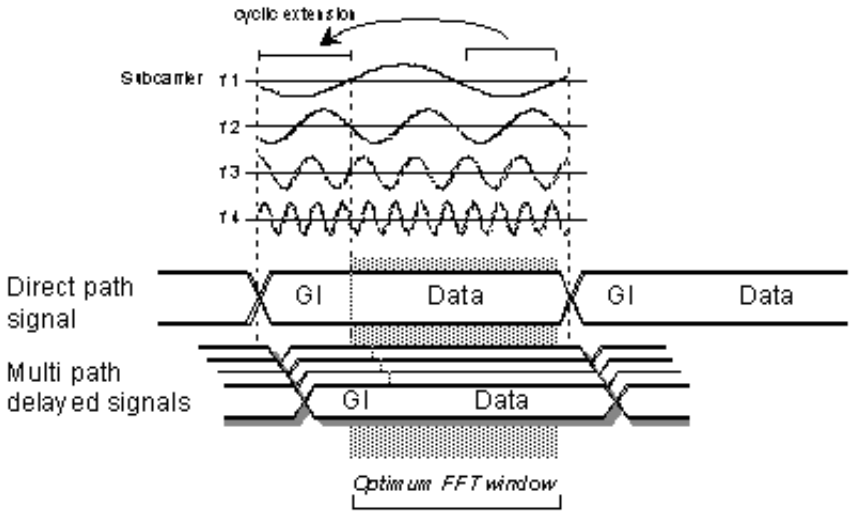

Figure 2.5: 802.11a OFDM symbol with cyclic extension.

802.11a is based on OFDM in the 5 GHz band. The basic principle of OFDM is to split a high rate data stream into a number of lower rate streams, which are transmitted simultaneously over a number of subcarriers. Since the symbol duration increases for the lower rate parallel subcarriers, the effects of time dispersion caused by multipath delay spread are decreased. Intersymbol interference is eliminated almost completely by introducing a guard time in every OFDM symbol. In the guard time, the OFDM symbol is cyclically extended, which introduces a large margin with respect to the transition of two successive symbols affected by multipath delay. Figure 2.5 shows an example of four subcarriers from one OFDM symbol. In practice, the most efficient way to generate the sum of a large number of subcarriers is by using the Inverse Fast Fourier Transform (IFFT). At the receiver side, the FFT can be used to demodulate all subcarriers. All subcarriers differ by an integer number of cycles within the FFT integration time, which ensures the orthogonality between the different subcarriers. This orthogonality is maintained in the presence of multipath delay spread, as illustrated in Figure 2.5. Because of multipath, the receiver sees a

Parameter	Value
Data Rate	6, 9, 12, 18, 24, 36, 48, 54 Mb/s
Modulation	BPSK, QPSK, 16-QAM, 64-QAM
Coding Rate	1/2, 2/3, 3/4
Number of subcarriers	52
Number of pilots	4
OFDM symbol duration	4 μs
Guard interval	800 ns
Subcarrier spacing	312.5 kHz
-3 dB Bandwidth	16.56 MHz
Channel spacing	20 MHz

Table 2.1: Main Parameters of the 802.11a Standard.

summation of time-shifted replicas of each OFDM symbol. As long as the delay spread is small compared to the guard time, there is no intersymbol interference neither intercarrier interference within the FFT interval of an OFDM symbol. The only remaining effect of multipath is a random phase and amplitude of each subcarrier. In order to deal with weak subcarriers in deep fades, forward error correction across the subcarriers is applied.

2.3.1 802.11a OFDM Parameters

Table 2.1 lists the main parameters of the 802.11a PHY standard. A key parameter, which largely determined the choice of the other parameters, is the guard interval of 800 ns. This guard interval provides robustness to root-mean-squared delay spreads up to several hundreds of nanoseconds, depending on the coding rate and modulation used. In practice, this means that the modulation is robust enough to be used in any indoor environment, including large factory buildings. It can also be used in outdoor environments, although directional antennas may be needed in this case for operation over larger distances.

In order to limit the relative amount of power and time spent on the guard time to 1 dB, the symbol duration was chosen to be 4 μs, including the guard interval. This also determined the subcarrier spacing to be 312.5 kHz, which is the inverse of the symbol duration minus the guard time. By using 48 data subcarriers, uncoded data rates of 12 to 72 Mb/s can be achieved by using variable modulation types from BPSK to 64-QAM. In addition to the 48 data subcarriers, each OFDM symbol contains an additional 4 pilot subcarriers, which can be used to track the residual carrier frequency offset, which remains after an initial frequency correction during the training phase of the packet.

In order to correct for subcarriers in deep fades, forward error correction across the subcarriers is used with coding rates of 1/2, 2/3, and 3/4, giving coded data rates from 6 up to 54 Mb/s.

2.4 Auto Rate Fallback

Different data rates have their own benefit in wireless communications. A WLAN network card communicating at, for example, 24 Mb/s cannot be said to be inferior to the one that communicates at 54 Mb/s, or vice versa. This is because lower data rates allow

communication possibility in worse channel conditions and over larger distances. Thus an Auto Rate Fallback (ARF) algorithm which can determine the best usable data rate for a given channel condition can improve the performance. This ARF algorithm is not defined in the 802.11 standard.

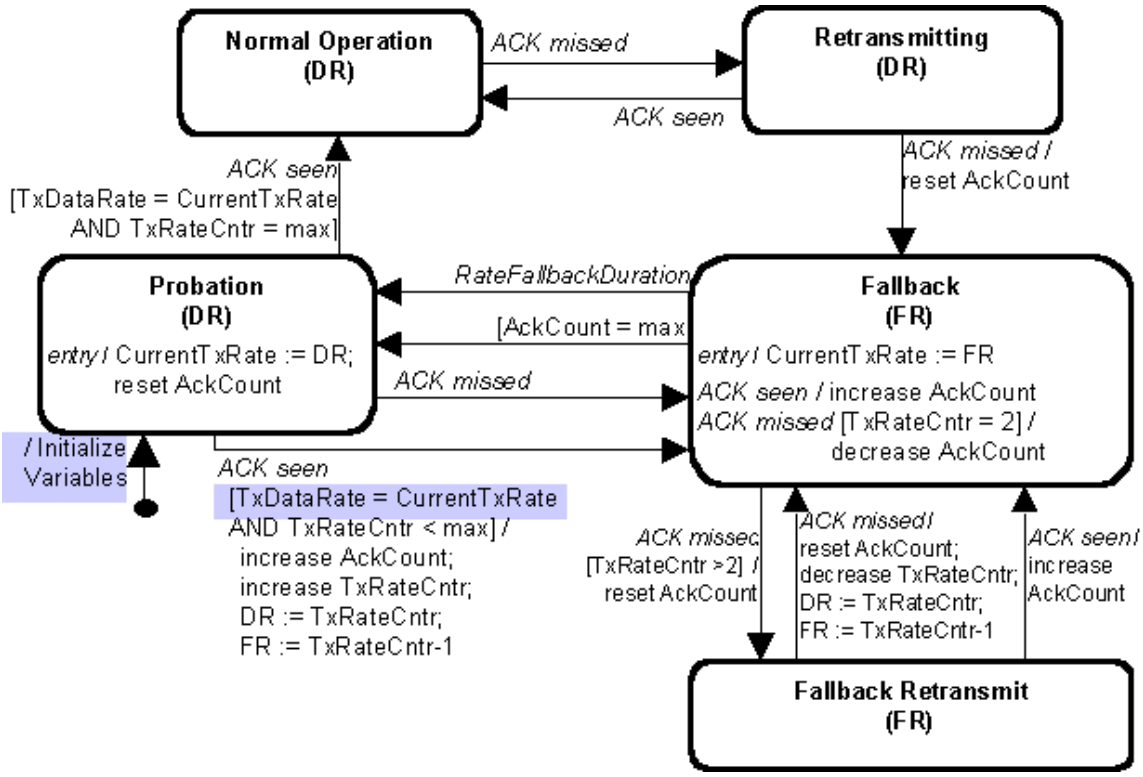

Figure 2.6: Agere Systems' Auto Rate Fallback.

Figure 2.6 shows an example of a Finite State Machine of the ARF algorithm as used by Agere Systems [11]. This Finite State Machine was developed for their 802.11b product. Starting from the Default Rate (DR), the fallback to the Fallback Rate (FR) is triggered by lost 802.11a ACKs. The fall forward to DR is controlled by attempts triggered by a timer or a number of successful acknowledged transmissions. Under any condition, the value of TxRateCntr is always the DR and TxRateCntr-1 is always the FR. Thus, the minimum value that TxRateCntr can have is 2.

There are five possible states:

Normal Operation (DR) indicates normal operation at the highest common data rate which is negotiated between the AP and the Mobile Terminal (MT). As long as each transmitted message receives an ACK, the MT stays in this state.

Retransmitting (DR) is entered for each initial retransmit attempt after an ACK is missed. In this state, the first retransmission is done at DR.

Fallback (FR) represents the operation at the FR. Initially FR is the second highest common data rate. If performance degrades, the values of FR and DR change. This state is entered when performance is degraded (based on missed ACK observations) while

operating at DR, or when ACK is missed or seen while in the state Fallback Retransmit. In this state, a count of ACKs is kept and an "upgrade timer" is running. When a certain time is spent in this state (controlled by configuration entity RateFallbackDuration) or when a certain number of ACKs have been received (controlled by configuration entity RateFallbackACKCount), an attempt is done to upgrade to DR, via the "Probation" state.

Fallback Retransmit (FR) delays the transition to a new FR value by two missed ACKs. This state is entered if ACK is missed while in Fallback state and TxRateCntr is greater than the minimum value.

Probation (DR) is "on the way back" to the normal DR operation. It involves one message transmission at DR. Receipt of an ACK returns to state "Normal Operation", a missed ACK returns to state "Fallback".

Chapter 3

Simulation Environments

The following simulation environments can be used for WLAN simulation:

- *ns*2 [13]
- OMNeT++ [14]
- MLDesigner [15]
- OPNET Modeler [16]

At the start of the project, the mentioned environments did not have an implementation of the IEEE 802.11a standard, except for OPNET Modeler. Therefore, we chose for OPNET Modeler as the simulation environment to use.

OPNET Modeler is a comprehensive software environment for modelling, simulating, and analysing the performance of communications networks, computer systems and applications, and distributed systems. It suits to analyse both behaviour and performance of modelled systems by performing discrete event simulations. The OPNET Modeler/Radio is an extension to the OPNET Modeler by providing radio-related capabilities. The models are described in a hierarchically object structure. It uses a Proto-C language (C/C++ oriented) to specify the models. The package is available for both UNIX and Windows NT platforms.

OPNET is geared more toward exploring network-wide design issues and conducting research at the MAC layer and above (i.e. IP, TCP, and UDP) than for examining the physical behaviour of wireless links. However, the model used incorporates the basic 802.11a PHY characteristics.

3.1 Modelling in OPNET

The OPNET environment incorporates tools for all phases of a simulation study, including model design, simulation, data collection, and data analysis. Several OPNET editors represent these phases: Project Editor, Process Editor, Parameter Editors and Advanced Editors. These Editors break down the required modelling information in a manner that parallels the structure of actual network systems. Thus, the model-specification editors are organized in an essentially hierarchical way. Model specifications performed in the Project Editor rely on elements in the Node Editor; in turn, the Node Editor makes use of the

models defined in the Process Editor. The remaining editors are used to define various data models, typically tables of values, that are later referenced by process- or node-level models.

3.1.1 Project Editor

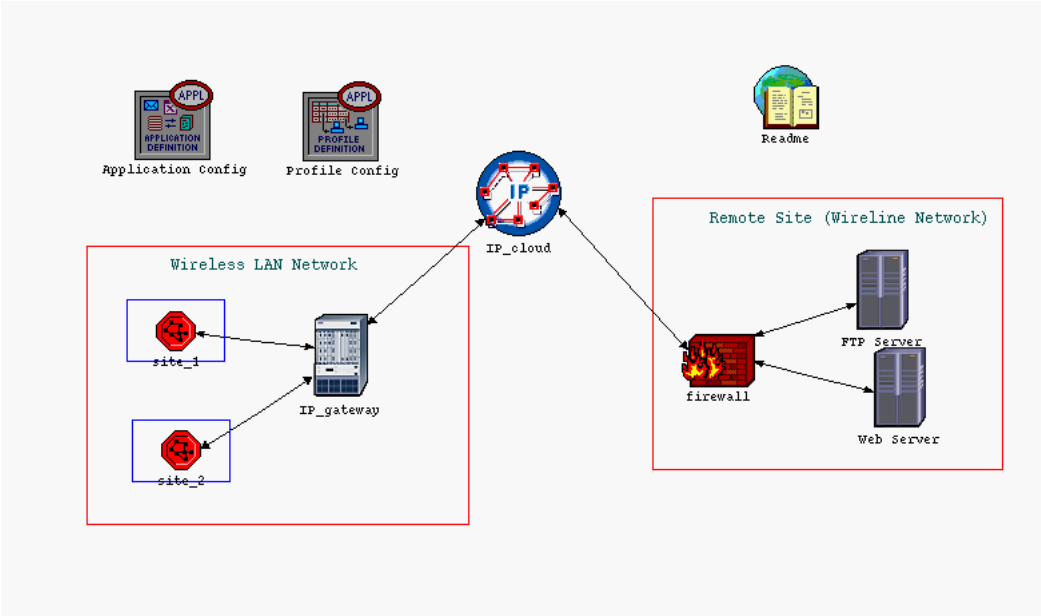

Figure 3.1: Project Editor.

This editor is used to develop network models. Network models are made up of subnets, node models and links. OPNET supports fixed, satellite and mobile nodes connected by point-to-point, bus or radio links. Subnetwork objects provide a hierarchy structure to breakdown the complexity of a network into several levels. This can be seen in Figure 3.1 with two subnetworks, one of these subnetworks is shown in Figure 3.2.

3.1.2 Node Editor

The Node Editor is used to define the behaviour of each network object (node models). Different modules define the internal aspects of the node models behaviour. Some of the modules have predefined behaviour (e.g. transmitters and receivers) while others are programmable. Modules are connected via packet streams or statistic wires. They allow the information to flow between the modules. Packet streams allow formatted messages called packets to be conveyed from one module to another. Statistic wires convey simple numeric signals or control information between modules.

This modelling paradigm is well suited to the modelling of “layered” or “stack” communication protocols. A module can represent a protocol layer of the protocol stack. Figure 3.3 represents the protocol stack of a Mobile Terminal (MT) used during the simulations. The Application, Transmission Control Protocol (TCP), Address Resolution Protocol (ARP), Internet Protocol (IP) and MAC-layers are all represented by a module. Sometimes it is necessary to use a number of modules to represent the complete functionality of a single

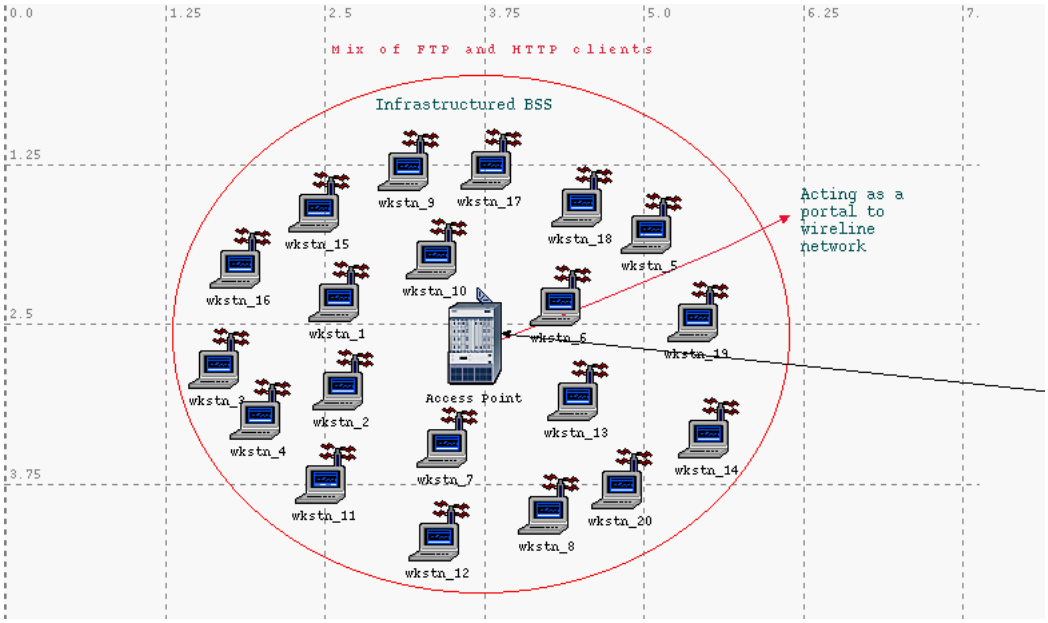

Figure 3.2: Site 1 of the Wireless LAN Network in Figure 3.1.

protocol. The IP protocol, for example, is subdivided into two separate processes: one for encapsulation/decapsulation and one for IP routing.

The interface with the Physical Layer, which will be covered in Section 3.2, is represented by eight transmitter modules and eight receiver modules. This makes it possible to model the eight different data rates, by assigning corresponding properties to each different transmitter–receiver pair.

3.1.3 Process Editor

The Process Editor is used to define the behaviour for the programmable modules. In this way, it is possible to control the underlying functionality of the node models created in the Node Editor. These models are used to simulate software subsystems, such as a communication protocol, and also to model hardware subsystems, such as the CPU of a MT.

A process is an instance of a process model and operates within one module. Initially, a process model contains only one process, this is referred to as “the root process”. However, a process can create additional “child processes” dynamically. These can in turn create additional processes themselves. This is well suited to model certain protocols.

Processes respond to interrupts. These interrupts indicate that events of interest have occurred like the arrival of a message or the expiration of a timer. An interrupted process takes actions in response to interrupts and then blocks, waiting for a new interrupt. It may also invoke another process and its execution is suspended until the invoked process blocks.

Finite state machines, named State Transition Diagrams (STDs) in OPNET, represent the process models. An example of a STD is shown in Figure 3.4. These STDs consist of icons representing states and lines that represent the transition between the states. The operations performed in each state or for a transition are expressed in Proto-C (embedded C/C++ code blocks and a library of Kernel Procedures providing commonly needed

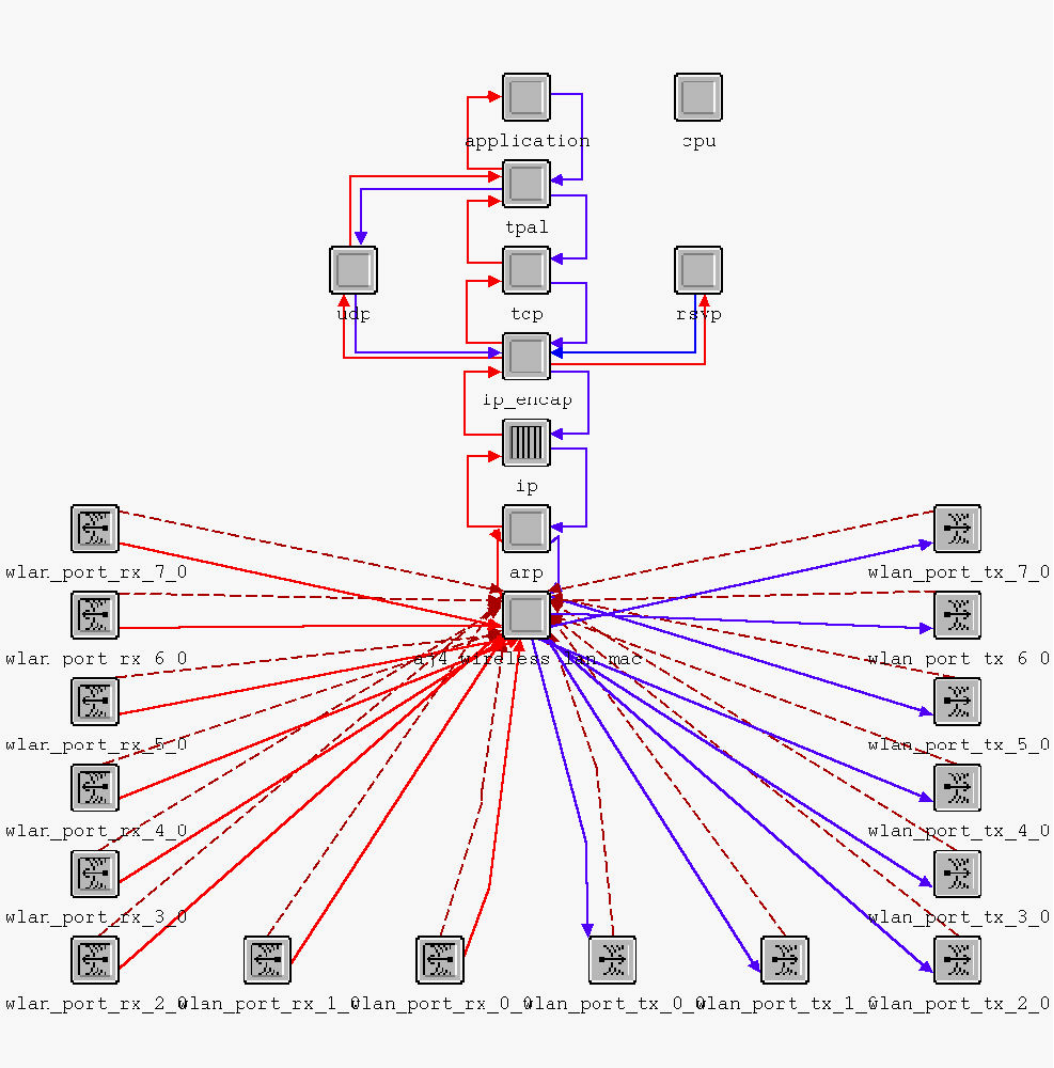

Figure 3.3: Workstation node model.

functionality for modelling communications and information processing).

The main features of a STD are:

The Initial State is the first state the process model enters upon invocation. This state is easily identified by a large arrow on its left-hand side (i.e. the INIT state in Figure 3.4). It usually performs functions such as the initialisation of variables.

The Transition Arc describes the possible movement of a process from one state to another and the conditions under which such a change in state may take place. A transition with no attached condition is depicted with a directed solid line, while one with an attached condition is depicted using a directed dashed line.

Transition Conditions are specified as Booleans. If no possible transition or more than one possible transition exists then the simulation halts. A "default transition" ensures that a situation where a simulation halts due to the fact that no transition evaluated to TRUE never occurs.

The Transition Executive is carried out when a transition is taken. As a transition is made from one state to another, actions can be executed when leaving the first state (exit executives) and upon entering the next state (enter executives).

Unforced States represent true states of a system. A process blocks after the enter executives of an unforced state have been executed. The exit executives are executed when a new interrupt causes the process to be re-invoked. The unforced states represent the possible stable states of a process. These states have a red colour in the process editor.

Forced States do not allow a process to wait or block. When a transition is followed that leads to a forced state, the enter executives are executed, immediately afterwards the exit executives are executed and another transition is followed. This chain continues until finally an unforced state is entered. Forced states are useful when attempting to simplify a complex task by subdividing the task into multiple forced states. The forced states are easily discriminated from the unforced states by its green colour.

Variables. OPNET processes not only include the facility to define variables for use during process invocations, "temporary variables", but also maintain a set of "state variables". While the values of the temporary variables are lost between process invocations, the values of state variables are maintained. State variables are typically used to model counters, statistical information and retransmission timer values while temporary variables are simply used to complete tasks such as packet handling.

State Attributes define a set of parameters, which can be used to tailor process instance behaviour. This allows generic specification of a process, which can be used in many different scenarios.

An example of the code found within the states is shown in Appendix B, where the exit executives of the "Wait for Response"-state are given. This is the first state on the right of Figure 3.4. Next to code hidden in the states, there are several other code blocks. These are the "State Variables", the "Temporary Variables", the "Header", the "Function", the "Diagnostic", and the "Termination" blocks. Appendix C gives an example of a function

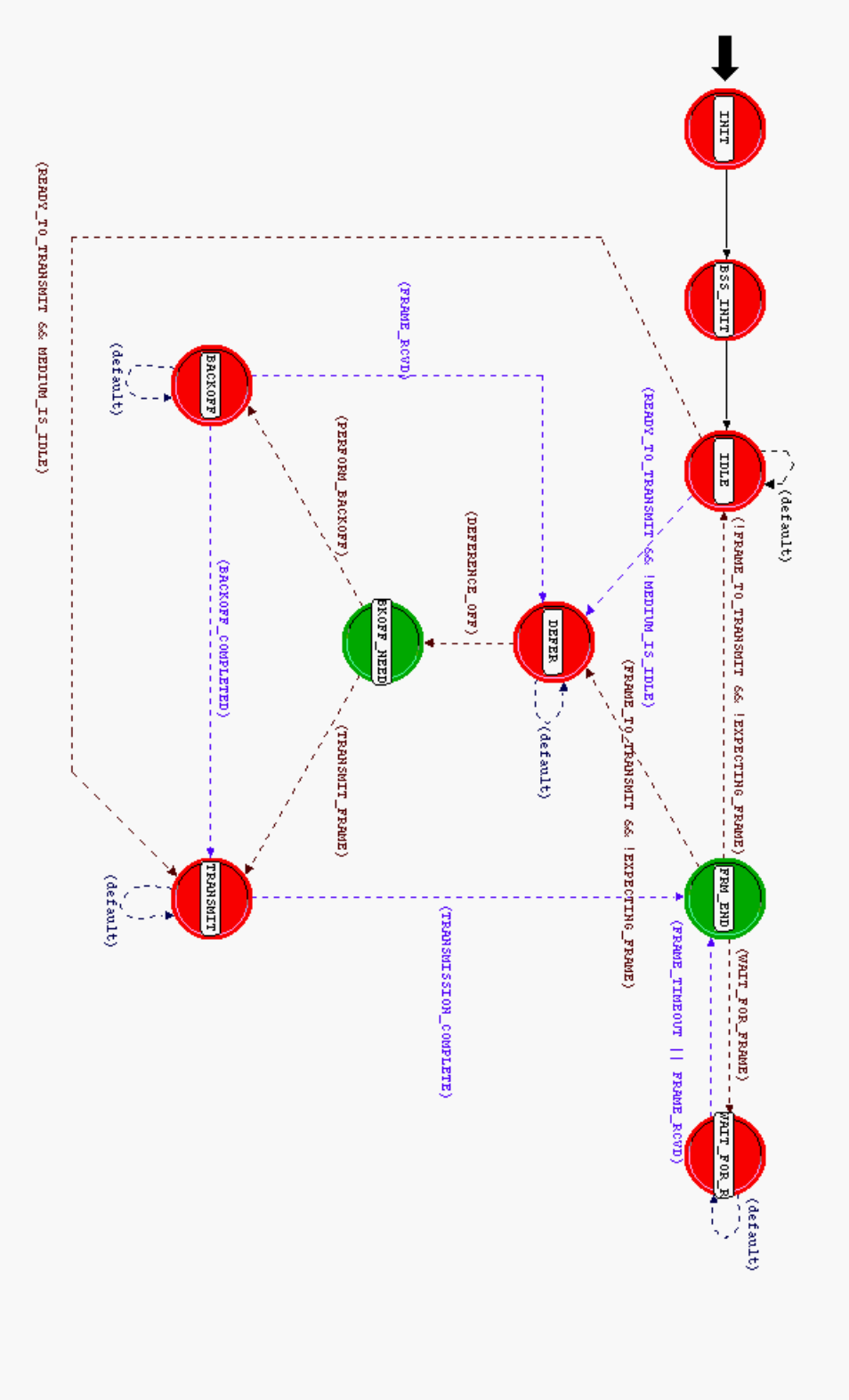

Figure 3.4: MAC layer Process Model.

defined in the Function Block. The total code for the MAC layer Process Model is about 5000 lines of code, and will, therefore, not be included in this report.

3.1.4 Parameter Editor

The Link Model Editor, Interface Control Information (ICI) Editor, Antenna Pattern Editor, Modulation Curve Editor and Probability Density Function (PDF) Editor are called the Parameter Editors and are used to specify a single type of models. These models take the form of a functional relationship or a table of information.

The Link Model Editor is used to create new types of link objects (link models). These link models can be used in the Network Editor to connect the nodes.

The Packet Format Editor is used to specify the collection of fields contained by a formatted packet.

The ICI Editor is used to define the internal structure of the ICIs. An ICI is a data structure that is used to establish a formal interface between modules that communicate via some form of interrupt.

The Antenna Pattern Editor is used to model the direction-dependent gain properties of antennas. It is used to characterise the effect of antenna patterns on the radio link performance. An example of a direction-dependent antenna gain is shown in Figure 3.5. During the simulations done in this report, we used an isotropic antenna pattern.

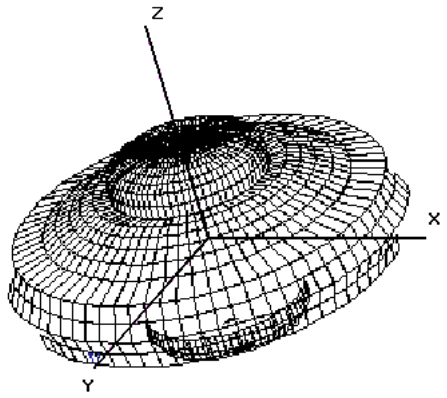

Figure 3.5: Three-dimensional antenna pattern.

The Modulation Curve Editor is used to create the mapping between the effective Signal-to-Noise Ratio (SNR) measured on a radio link and the expected value of the Bit Error Rate (BER) that would result. In this way, it is possible to characterise the vulnerability of an information coding and modulation scheme to noise. More details on this editor can be found in paragraph 3.2.2.

The PDF Editor is used to map real values of a random variable to their associated probability densities.

3.1.5 Advanced Editors

The Probe Editor, Simulation Editor, and Analysis Editor are all advanced versions of those that are available in the Project Editor.

The Probe Editor is used to specify the statistics to be collected during simulation. Although this can be done from the Project Editor, it is possible to specify advanced characteristics for each probe in the Probe Editor. There are several different types of statistics that can be collected using different probes, including global statistics, link statistics, node statistics, attribute statistics, and several types of animation statistics.

The Simulation Editor is used to specify sequences that control the simulations. While this can be done in the Project Editor, the Simulation Editor can be used to specify additional simulation constraints.

The Analysis Editor is used to view the simulation results. The advances above the Project Editor version are features like the creation of templates and the creation of analysis configurations.

3.2 The Physical Layer Model

OPNET models the physical layer by a 14 stage pipeline as shown in Figure 3.7. Each stage is defined to model a particular aspect of link performance. Pipeline stages typically consist of formulas and algorithms applied to transmission data.

3.2.1 The Transceiver Pipeline Stages

The pipeline is divided into two groups, one associated with the transmitter, and one associated with the Receiver:

0. Receiver Group: determines feasibility of communication, this stage is not executed on a per transmission basis.

Transmitter Associated Stages:

1. Transmission Delay Model: computes the time required for a transmission to complete.
2. Link Closure Model: determines which receivers can be reached by the transmission (the rest of the Pipeline will be executed separately for each receiver that passes this test).
3. Channel Match Model: determines which receiver channels can demodulate the transmission, and which ones should treat it as noise.
4. Transmitter Antenna Gain Model: computes the gain of the transmitter's antenna in the direction of the receiver.
5. Propagation Delay Model: computes the time required for the transmitted signal to propagate between the transmitter and the receiver.

Receiver Associated Stages:

6. Receiver Antenna Gain Model: computes the gain of the receiver's antenna in the direction of the transmitter.

7. Received Power Model: computes the received power level (typically factoring in antenna gains, channel frequency, transmitter power, distance, etc.). Fading effects can be incorporated here.

   We have implemented the following Path Loss model:

$$P_l = \begin{cases} \frac{\lambda^2}{16\pi^2 r^2}, & 0 < r < 5.0\ m \\ \frac{\lambda^2}{16\pi^2 r^{3.3}} \cdot \frac{5^{3.3}}{5^2}, & r \geq 5.0\ m \end{cases}$$

   where P_l is the path loss, λ is the wavelength in meters, and r is the distance from the transmitting antenna in meters. This path loss model has a breakpoint at 5 meter, with free space path loss up to 5 meter ($\sim 1/r^2$), and after 5 meter more loss ($\sim 1/r^{3.3}$), corresponding to Agere Systems' path loss model for the open office environment [7].

   To give an impression of the code needed for a stage, the code for this stage is given in Appendix D.

8. Background Noise Model: computes the in-band background noise based on the receiver noise factor.

9. Interference Noise Model: computes the interference noise which affects a transmission fragment (typically, the total power of all other concurrent, in-band transmissions). This stage is only invoked if packet overlap occurs and divides the packets into portions of constant SNR as shown in Figure 3.6.

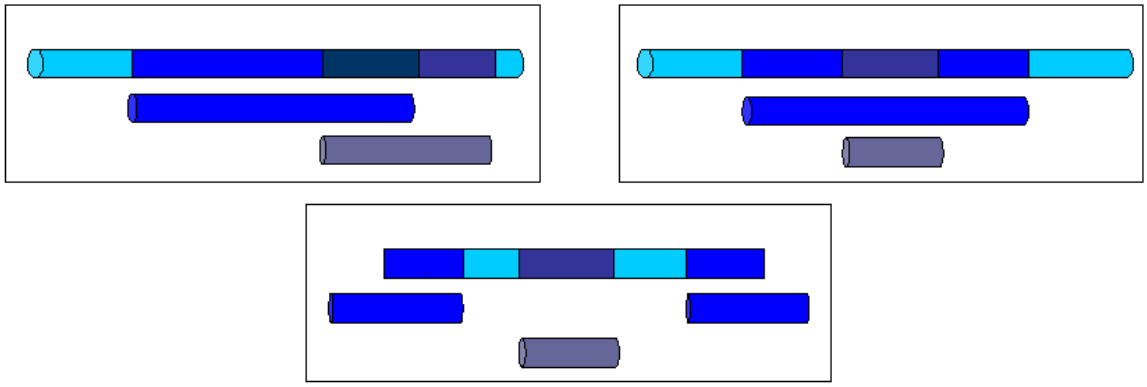

Figure 3.6: Examples of SNR Variations.

10. SNR Model: computes the SNR of a transmission fragment.

$$SNR = 10 \cdot \log(\frac{P_r}{P_b + P_i})$$

    Where P_r is the Received Power, P_b is the Background Noise, and P_i is the Interference Noise. The Interference Noise is assumed to be Gaussian.

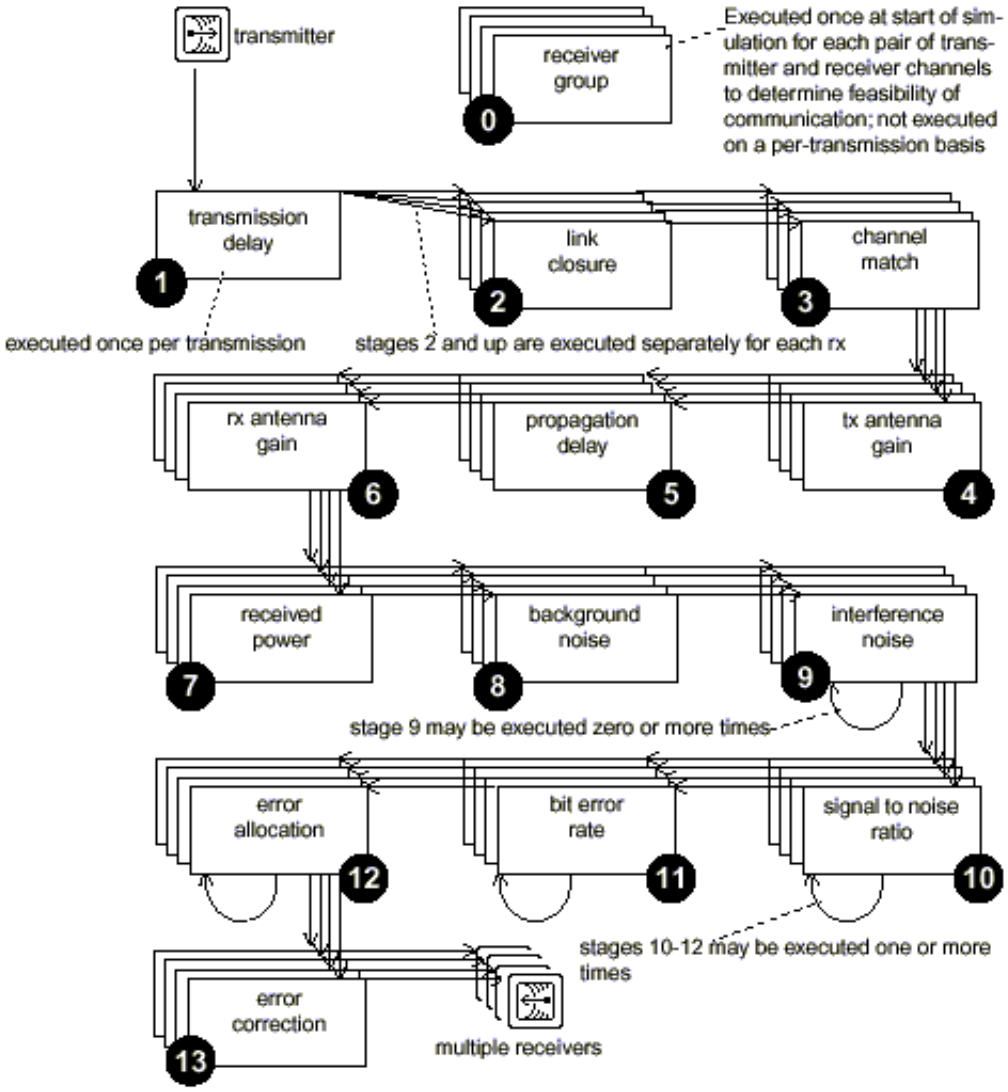

Figure 3.7: Pipeline stages.

11. BER Model: computes the mean bit error rate (BER) over each constant SNR fragment of the transmission. This stage uses a modulation table to determine the BER, this process is discussed in more detail in Section 3.2.2.

12. Error Allocation Model: determines the number of bit errors in each fragment of the transmission.

13. Error Correction Model: determines whether the allocated transmission errors can be corrected and if the transmitted data should be forwarded into the node for higher level processing. This is done by providing an Error Correction Code (ECC) threshold. The ECC threshold is the highest proportion of bit errors allowed in a packet in order for the packet to be accepted by a receiver and forwarded to the output stream.

    However, with our 802.11a PHY model and modulation tables for 6–54 Mb/s the effect of error correction is already included in the BER curves. Therefore, the ECC threshold is set to zero.

3.2.2 The Modulation Table

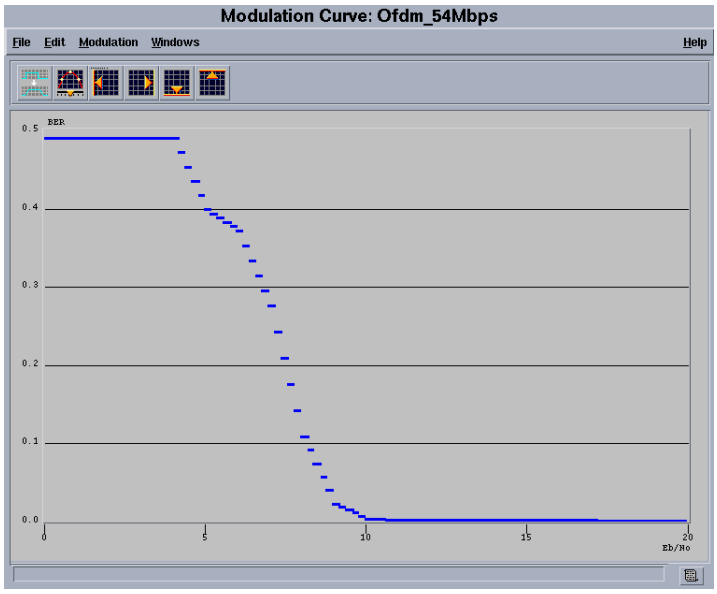

Figure 3.8: The Modulation Curve used in the 802.11a model for 54 Mb/s by Braswell.

In OPNET the Modulation Curves are defined using a graphical approach. The BER axis scale of the modulation function graph is linear, and therefore not logarithmic as is more usual to represent BER curves. This results in an inaccurate definition of the modulation curve for the most relevant BER values (BER 10^{-2}–10^{-8}), as can be seen in Figure 3.8.

The OPNET Modulation Curve Editor displays the horizontal axes as E_b/N_o, the Energy per Bit divided by the Noise per unit of bandwidth. When referring to the code of the pipeline stage, it shows that actually the SNR value is used to lookup the BER-value in the table. The relation between SNR and E_b/N_o is given by the following equation [10]:

$$SNR = \frac{R_b}{W} \cdot \frac{E_b}{N_o}$$

where E_b is the signal energy per bit, R_b is the bit transmission rate, N_o the power spectral density of the additive white Gaussian noise (an indicator of the noise power), while W is the bandwidth. The fact that OPNET doesn't make a distinction between SNR and E_b/N_o is no problem as long the bandwidth efficiency $R_b/W = 1$ bit/sec/Hz. As this is not the case for the different data rates within 802.11a, we decided to use the BER versus SNR curves found in reference [8] after correction for expected system degradation (for better correspondence to practical BER behaviour) and shown in Figure 3.9.

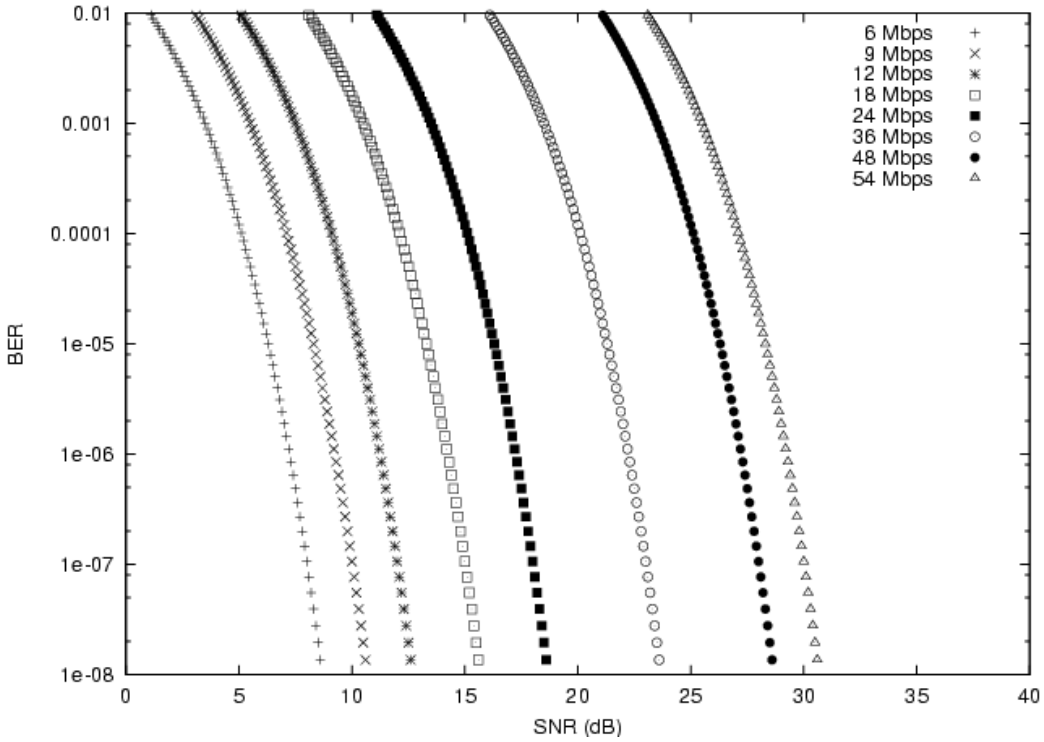

Figure 3.9: The BER-curves used in the simulations.

The most precise way to specify the modulation curve is through EMA (External Model Access) code. This is a technique of accessing a model external to the OPNET program (i.e., without using the services provided by the OPNET graphical editors). Using this technique, we were able to implement the curves shown in Figure 3.9 in OPNET and use these curves during simulations.

Chapter 4

Simulations

The following simulations were done using a model based on the IEEE 802.11a model made by dr Sunghyun Choi [5] of Philips Research Labs. This model was later adjusted by Bryan Braswell [6]. The model is attainable as a contributed model at the OPNET website.

We started with some simple simulations to compare the results with regard to earlier work, based on 802.11b throughput evaluation. The quantative analysis of OPNET results compared to measurements indicates that the results are realistic, see "Interpretation of OPNET results" [12].

There are many applications of which the traffic can be simulated by OPNET, for example E-mail, file transfer, video conferencing, voice over IP and web browsing. We chose for File Transfer (FTP) traffic in our simulations. Traffic can be configured using the "Application Config" and "Profile Config" icons shown in Figure 4.1.

4.1 Coverage Range of 802.11a

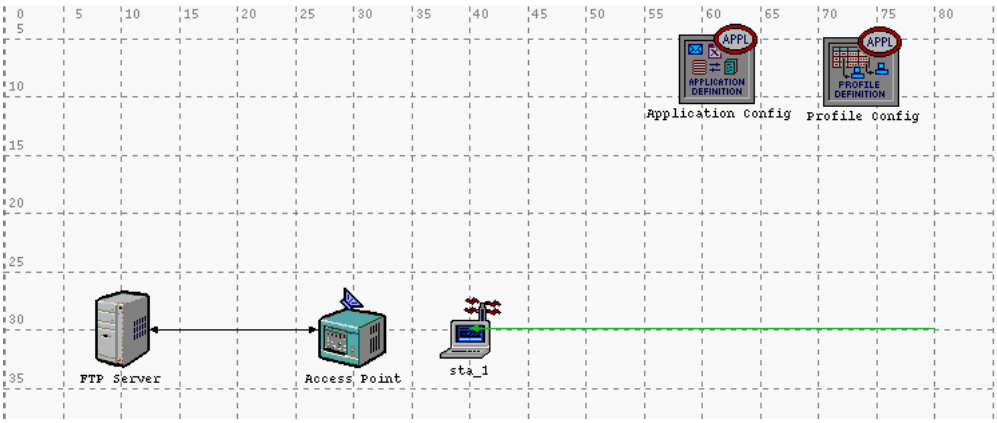

Figure 4.1: Simulation scenario.

To determine the coverage range of each 802.11a data rate, simulations were done for the scenario shown in Figure 4.1. The grid shown in this figure gives the position of the nodes expressed in meters. The simulations were conducted using an infrastructure BSS with a fixed AP and a single MT. The MT moves during the simulation, increasing the

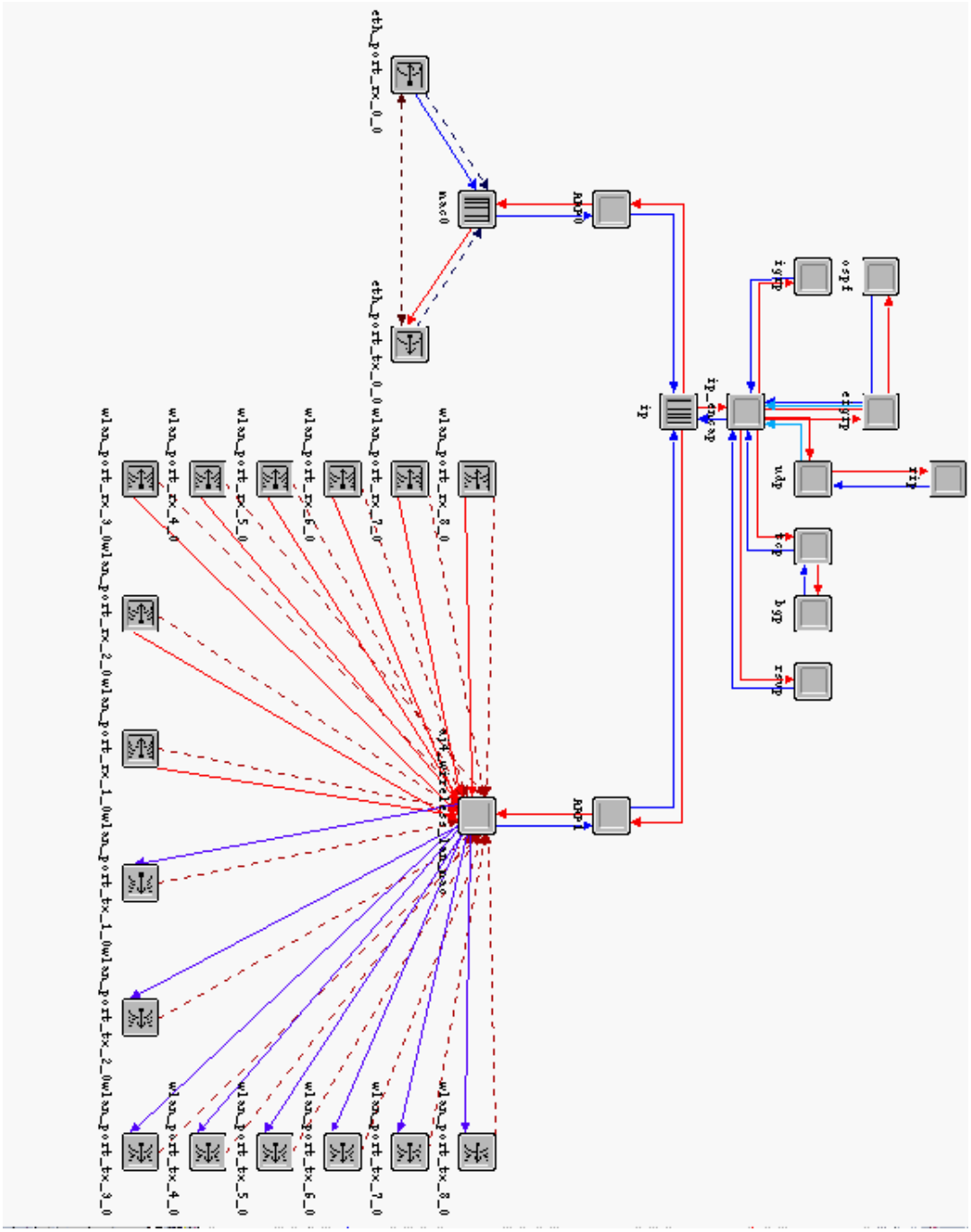

Figure 4.2: Access Point node model.

distance between the AP and the MT. As a result, the SNR will deteriorate resulting in a link failure.

The node model of the MT is given in Figure 3.3, the node model of the AP is given in Figure 4.2. The AP node model has two interfaces, one for a wired network and another for the WLAN.

The FTP Server is connected to the AP using Gigabit Ethernet, therefore the wired part of the scenario will not limit throughput results during simulations. During the simulation, the MT downloads every second a 3 Mbyte file from the FTP Server. It takes the MT 10 minutes to move from position (40, 30) to position (80, 30). Therefore, the speed of the MT is 40 meters per 10 minutes.

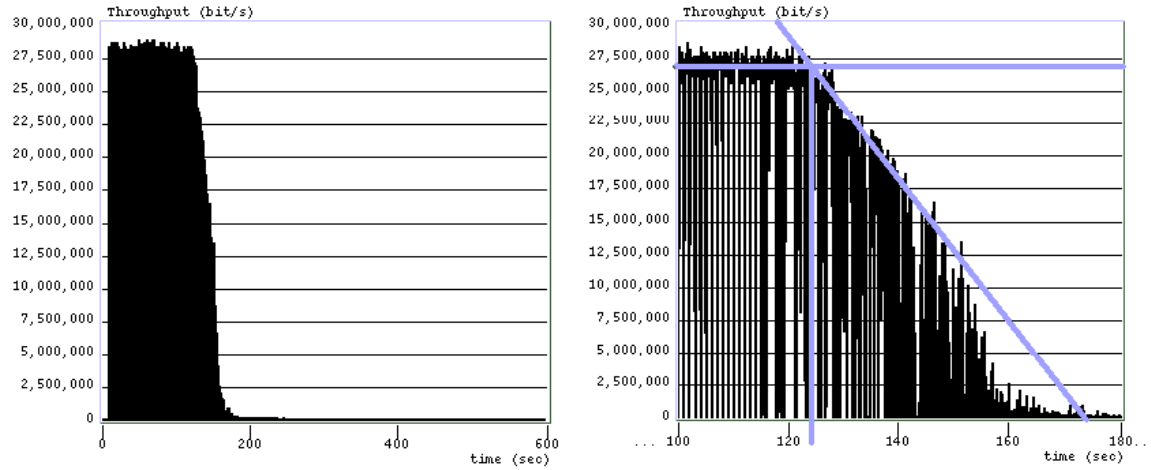

Figure 4.3: Throughput with data rate set at 54 Mb/s for the full 10 minutes of simulation time on the left, and for the period from 100 to 180 seconds on the right.

The left graph in Figure 4.3 shows the throughput during the 10 minutes of the simulation when the data rate is set at 54 Mb/s. The actual net throughput is shown to be around 27.5 Mb/s, this is a result of overhead. After approximately 120 seconds, the throughput starts to decrease until, at 180 seconds, the throughput stays at zero for the rest of the simulation. The graph on the right in Figure 4.3 shows the throughput during the period from 100 to 180 seconds after the start of the simulation. This graph shows that at 124 seconds the throughput starts to drop as a result of a deteriorating SNR. After 124 seconds the MT has moved $124 \cdot 40/(10 \cdot 60) \approx 8$ m. The 8 meters plus the off-set of 10 meters distance between the AP and the MT at the start of the simulation result in a total coverage range of 18 meters for the 54 Mb/s data rate. These simulations have been done for all eight different data rates, the results are given in the last column of Table 4.1.

The Table 4.1 also gives a calculated coverage range. The simulation doesn't take fading into account, therefore the simulation coverage range should be the same as the calculated coverage range with a fading margin equal to zero. This is not the case; there is a difference between the results of the simulation and the calculated values equivalent to a fading margin of 2.1 dB. This might be a result of, among others, a different criterion for the coverage range, and interpolation of the BER table by OPNET (defined in steps of 0.1 dB). When, for example, in the 54 Mb/s case in Figure 4.3 the value of 160 seconds is taken, instead of the 124 seconds, the range would be 21 meters. As the relative distances and not the

Data Rate (Mb/s)	Calculated range (m) Fading Margin = 0 dB	Calculated range (m) Fading Margin = 2.1 dB	Range (m) Simulation
6	100	87	86 ±1
9	87	75	75 ±1
12	76	66	64 ±1
18	62	53	51 ±1
24	50	43	42 ±1
36	35	30	29 ±1
48	25	21	21 ±1
54	22	19	18 ±1

Table 4.1: 802.11a Coverage Ranges.

absolute distances are of importance when looking at the Auto Rate Fallback, the settings leading to the results in Table 4.1 are used during the simulations in the following sections.

4.2 Auto Rate Fallback

Auto Rate Fallback can be implemented as outlined in Section 2.4, based on missed 802.11a ACKs. Other examples are Auto Rate Fallback based on:

Link SNR The SNR as measured at the receiver is a good indicator of the quality of a wireless data link. This information is not available at the transmitter side and therefore this option is not realistic. Furthermore, there is no static situation. The interference comes in short bursts, which cannot be predicted.

Frame Loss Rates The number of frames dropped during the transmission process also serves as an indicator of link performance in a WLAN. This option is less flexible than using missed ACKs as it takes multiple missed ACKs to have a frame dropped. Again, bursty conditions make the average frame loss rate provide no relevant information.

We chose to use the missed ACKs to trigger our Auto Rate Fallback algorithm. A flexible way of incorporating different Auto Rate Fallback algorithms was chosen by using a table. Every position in this table represents a certain "state" and contains three values. The first value is the current data rate, the second is the position to jump to within the table when receiving an ACK, and the third value is the position to jump to when missing an ACK. The three tables used during simulations are shown in Appendix A.

Table 1 needs 4 missed ACKs in a row before fallback and needs subsequently 5 seen ACKs in a row before recovering. This table has no probation state.

Table 2 is the same as table 1, but this table does have a probation state. Therefore, after 5 seen ACKs, the probation state is entered. In this probation state, there are two possibilities. If the next ACK is missed, the fallback state is entered again. If the next ACK is seen, the system will recover.

Table 3 is the same as table 2, but now it takes 11 seen ACKs to recover.

There are several advantages of the table-implementation of the algorithm. Every data rate can have a different way of reacting to missed ACKs. The algorithm can be easily adjusted. Less risk of an error in the implementation, as the table gives a good overview of the algorithm. In addition, the history of missed ACKs is built into the table. It is noted that a final implementation could be based on another design approach, but with similar behaviour.

The ARF algorithm has to be optimised in different scenarios. An ACK can be missed because the receiver didn't receive the packet. This can be a result of a collision or of a bad SNR. For both situations, a scenario has been designed to test the ARF algorithm.

4.2.1 The Link Failure Scenario

This scenario is the same as in Figure 4.1. In this case, the MT, after reaching the most distant position from the AP, moves back towards the AP. In this scenario, the distance between the AP and MT at the start of the simulation is 15 m. In addition, the code needed for the Auto Rate Fallback Algorithm at the AP and the MT is added. During the first 10 minutes of the simulation, the algorithm will lower the operational data rate as the radio link quality deteriorates. During the last 10 minutes of the simulation, the operational data rate will increase as can be seen in Figure 4.4. This simulation has been done for the three different tables shown in Appendix A. The resulting throughput is plotted in Figure 4.5.

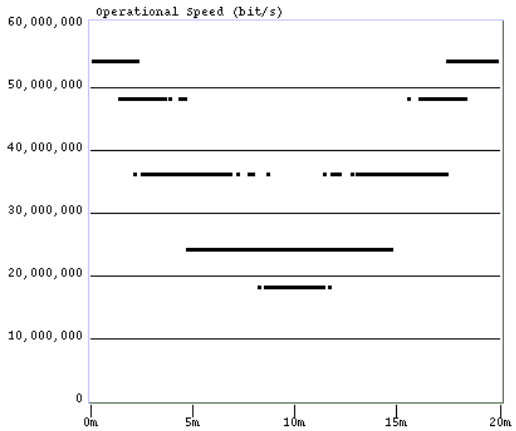

Figure 4.4: Operational Data Rate (with horizontally the time in minutes).

The results show that the lack of a Probation State in table 1 results in a significantly lower throughput once the 54 Mbit/s range is left by the MT. Furthermore the table 3 throughput is higher than the table 2 throughput. This is a result of the fact that table 3 needs 11 seen ACKs before recovering to a higher data rate instead of 5. As table 3 had the best throughput results in this scenario, this table will be used to test in the collision scenario in the next paragraph.

4.2.2 The Collision Scenario

To test how the Auto Rate Fallback algorithm will react on missed ACKs as a result of collisions the scenario shown in Figure 4.6 was used. After 10 seconds all of the 16 MTs start uploading a 1 Mbyte file to the FTP Server.

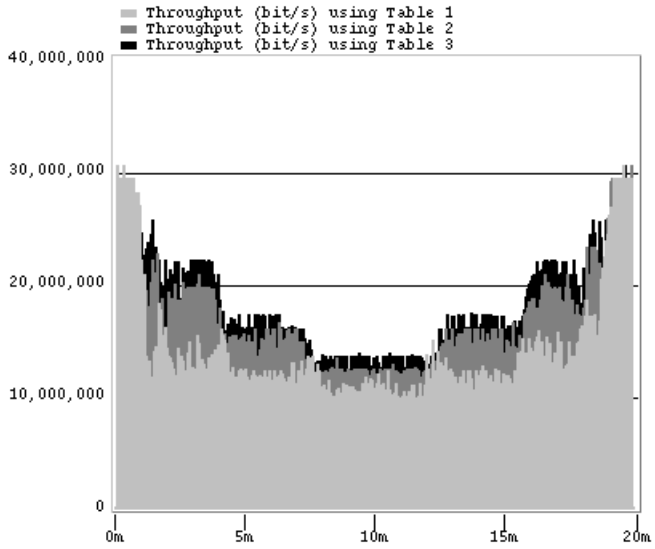

Figure 4.5: Throughput using Table 1, 2, and 3 (with horizontally the time in minutes)

The results show that the operational speed of the MTs is almost continuously 54 Mb/s, and sporadic 48 Mb/s. This algorithm performs well under these conditions, as there is hardly any fallback. Therefore, it can be concluded that 802.11a ACKs missed due to collisions is an uncommon phenomenon. This probably is a result of the TCP protocol shaping the traffic, as the offered traffic load at the transmitter side is influenced by the TCP protocol.

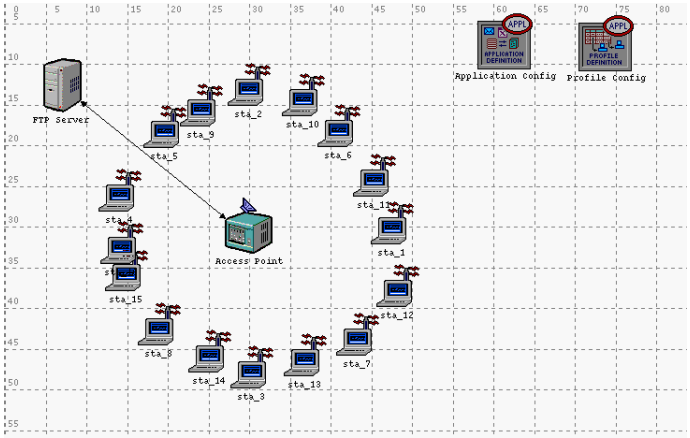

Figure 4.6: Collision scenario.

4.2.3 The Multiple APs Scenario

A third scenario was designed to investigate the impact of channel re-use. What would be the impact of a second AP and multiple MTs transmitting on the same, or a different but partly overlaying channel in the neighbourhood of the original AP? Unfortunately, the OPNET MAC-layer code demonstrated to be incapable of operating in this scenario.

Chapter 5

Conclusions

The goal of this report is two-fold; the first goal is to look at Auto Rate Fallback for the IEEE 802.11a protocol, the second goal is to evaluate OPNET Modeler as a WLAN simulation environment. Conclusions concerning these two goals are given in the following two paragraphs. The final paragraph includes recommendations for further research.

5.1 Auto Rate Fallback Algorithm

The first conclusion that can be drawn from the simulation results of the "Link Failure Scenario" is that a so called "Probation State" within the Auto Rate Fallback algorithm is essential to have a good throughput performance. This is a result of the fact that it takes 4 missed 802.11a ACKs instead of 1 missed ACK to fallback after recovery.

Secondly, the results show that a "hard recovery" gives better throughput results. In other words, the more seen ACKs are needed before trying to transmit at a higher data rate, the better the performance in the "Link Failure Scenario". The ratio between the number of ACKs seen and the 1 ACK missed in the "Probation State" is higher and therefore gives a higher throughput.

The simulations done with 16 Mobile Terminals uploading simultaneously a 1 Mbyte file lead to a very high traffic load in this "Collision Scenario". Even under these extreme conditions, fallback from 54 Mb/s to 48 Mb/s is extremely rare. Therefore, it can be concluded that 802.11a ACKs missed due to collisions is an uncommon phenomenon. This is probably a result of the TCP protocol shaping the traffic.

Concrete, this leads to the conclusion that a Auto Rate Fallback algorithm with easy fallback (i.e. 4 missed ACKs) and hard recovery (i.e. 11 seen ACKs) will give the best throughput results for both scenarios.

For other scenarios and practical implementation, the retry rate will be an important factor for the settings in relation to the number of ACKs missed before fallback and ACKs seen before fallforward.

5.2 Evaluation of OPNET Modeler

OPNET is geared more towards exploring network-wide design issues and conducting research at the MAC layer and above (i.e. IP, TCP, and UDP), than for examining the physical behaviour of wireless links. The way the physical layer is modelled, by serial stages, is very

limiting. OPNET is, for example, not capable of simulating an office environment; creating walls and different floors is not possible.

OPNET is not designed for WLAN simulation. Having multiple BSSs interfering with each other is not possible due to limitations in the detail of the model.

5.3 For Further Research

The flexible way of implementing the Auto Rate Fallback algorithm makes it possible to test all kinds of different algorithms. By introducing a certain percentage of extra faults, the robustness of the algorithm can be studied. This could give interesting results regarding the extra faults introduced in Agere Systems' product with Wireless Encryption Protocol.

Furthermore, extending the OPNET code of the MAC-layer would make it possible to have multiple BSSs to hear each other. This would extend the simulation possibilities significantly in relation to multicell/multichannel interference, as occurring in a large office building. However, this would be a very time consuming effort.

Bibliography

[1] Institute of Electrical and Electronics Engineers, 802.11, *"Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications"*, 20 August 1999.

[2] Institute of Electrical and Electronics Engineers, 802.11b, *"Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band"*, 16 September 1999.

[3] Institute of Electrical and Electronics Engineers, 802.11a, *"Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-Speed Physical Layer in the 5 GHz Band"*, 16 September 1999.

[4] ETSI TR 101 683 V1.1.1, *"Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; System Overview"*, February 2000.

[5] Choi, S. (Philips Research Labs, New York), OPNET Model of the 802.11a Protocol, 15 November 2000.

[6] Bryan E. Braswell (Naval Postgraduate School, Monterey , California), *"Modeling Data Rate Agility in the IEEE 802.11a Wireless Local Area Networking Protocol"*, March 2001.

[7] Prasad, Anand R., and Prasad, N. *"WLAN Systems and Wireless IP for Next Generation Communications"*, Artech House, ISBN 1-58053-290-X.

[8] Nee, R. van, and R. Prasad *"OFDM for Wireless Multimedia Communications,"* Artech House universal personal communications library, ISBN 0-89006-530-6

[9] OPNET, *"OPNET Modeler User Manuals"*

[10] Pahlavan, Kavah, and Allen H. Levesque *"Wireless Information Networks,"* Wiley series in telecommunications and signal processing, ISBN 0-471-10607-0.

[11] Prasad, Anand R., and Henri Moelard *"WaveLAN-II System Design Note 225: Enhanced Data Rate Control,"* March 19, 1999.

[12] Kamerman, A. *"NTS Design Note: Interpretation of OPNET results"*, February 4, 2002.

[13] The Network Simulator - ns-2, *http://www.isi.edu/nsnam/ns/*

[14] OMNeT++, *http://www.hit.bme.hu/phd/vargaa/omnetpp.htm*

[15] MLDesign Technologies, *http://www.mldesigner.com/*

[16] OPNET Optimum Network Performance, *http://www.opnet.com*

Appendix A

Auto Rate Fallback Tables

In the "Header Block" of the OPNET process model for the IEEE 802.11a MAC, we defined the three tables given in this appendix. The tables are used to model the Auto Rate Fallback algorithm as described in section 4.2. Every position in these tables represents a certain "state" and contains three values. The first value is the current data rate, the second is the position to jump to within the table when receiving an ACK, and the third value is the position to jump to when missing an ACK.

Table 1 needs 4 missed ACKs in a row before fallback and needs subsequently 5 seen ACKs in a row before recovering. This table has no probation state.

Table 2 is the same as table 1, but this table does have a probation state. Therefore, after 5 seen ACKs, the probation state is entered. In this probation state, there are two possibilities. If the next ACK is missed, the fallback state is entered again. If the next ACK is seen, the system will recover.

Table 3 is the same as table 2, but now it takes 11 seen ACKs to recover.

position	Table 1			Table 2			Table 3		
	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	54	8	9	54	8	9	54	8	9
5	54	4	9	54	4	9	54	4	9
6	54	5	9	54	5	9	54	5	9
7	54	6	9	54	6	9	54	6	9
8	54	7	9	54	7	9	54	7	9
9	54	7	10	54	7	10	54	7	10
10	54	7	11	54	7	11	54	7	11
11	54	7	24	54	7	24	54	7	24
12	0	0	0	0	0	0	0	0	0

position	Table 1			Table 2			Table 3		
	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed
13	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	48	11	25
15	0	0	0	0	0	0	48	14	25
16	0	0	0	0	0	0	48	15	25
17	0	0	0	0	0	0	48	16	25
18	0	0	0	0	0	0	48	17	25
19	0	0	0	0	0	0	48	18	25
20	48	8	25	48	11	25	48	19	25
21	48	20	25	48	20	25	48	20	25
22	48	21	25	48	21	25	48	21	25
23	48	22	25	48	22	25	48	22	25
24	48	23	25	48	23	25	48	23	25
25	48	23	26	48	23	26	48	23	26
26	48	23	27	48	23	27	48	23	27
27	48	23	40	48	23	40	48	23	40
28	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	36	27	41
31	0	0	0	0	0	0	36	30	41
32	0	0	0	0	0	0	36	31	41
33	0	0	0	0	0	0	36	32	41
34	0	0	0	0	0	0	36	33	41
35	0	0	0	0	0	0	36	34	41
36	36	24	41	36	27	41	36	35	41
37	36	36	41	36	36	41	36	36	41
38	36	37	41	36	37	41	36	37	41
39	36	38	41	36	38	41	36	38	41
40	36	39	41	36	39	41	36	39	41
41	36	39	42	36	39	42	36	39	42
42	36	39	43	36	39	43	36	39	43
43	36	39	56	36	39	56	36	39	56
44	0	0	0	0	0	0	0	0	0

position	Table 1			Table 2			Table 3		
	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed
45	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	24	43	57
47	0	0	0	0	0	0	24	46	57
48	0	0	0	0	0	0	24	47	57
49	0	0	0	0	0	0	24	48	57
50	0	0	0	0	0	0	24	49	57
51	0	0	0	0	0	0	24	50	57
52	24	40	57	24	43	57	24	51	57
53	24	52	57	24	52	57	24	52	57
54	24	53	57	24	53	57	24	53	57
55	24	54	57	24	54	57	24	54	57
56	24	55	57	24	55	57	24	55	57
57	24	55	58	24	55	58	24	55	58
58	24	55	59	24	55	59	24	55	59
59	24	55	72	24	55	72	24	55	72
60	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0	0	0
62	0	0	0	0	0	0	18	59	73
63	0	0	0	0	0	0	18	62	73
64	0	0	0	0	0	0	18	63	73
65	0	0	0	0	0	0	18	64	73
66	0	0	0	0	0	0	18	65	73
67	0	0	0	0	0	0	18	66	73
68	18	56	73	18	59	73	18	67	73
69	18	68	73	18	68	73	18	68	73
70	18	69	73	18	69	73	18	69	73
71	18	70	73	18	70	73	18	70	73
72	18	71	73	18	71	73	18	71	73
73	18	71	74	18	71	74	18	71	74
74	18	71	75	18	71	75	18	71	75
75	18	71	88	18	71	88	18	71	88
76	0	0	0	0	0	0	0	0	0

position	Table 1			Table 2			Table 3		
	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed
77	0	0	0	0	0	0	0	0	0
78	0	0	0	0	0	0	12	75	89
79	0	0	0	0	0	0	12	78	89
80	0	0	0	0	0	0	12	79	89
81	0	0	0	0	0	0	12	80	89
82	0	0	0	0	0	0	12	81	89
83	0	0	0	0	0	0	12	82	89
84	12	72	89	12	75	89	12	83	89
85	12	84	89	12	84	89	12	84	89
86	12	85	89	12	85	89	12	85	89
87	12	86	89	12	86	89	12	86	89
88	12	87	89	12	87	89	12	87	89
89	12	87	90	12	87	90	12	87	90
90	12	87	91	12	87	91	12	87	91
91	12	87	104	12	87	104	12	87	104
92	0	0	0	0	0	0	0	0	0
93	0	0	0	0	0	0	0	0	0
94	0	0	0	0	0	0	9	91	105
95	0	0	0	0	0	0	9	94	105
96	0	0	0	0	0	0	9	95	105
97	0	0	0	0	0	0	9	96	105
98	0	0	0	0	0	0	9	97	105
99	0	0	0	0	0	0	9	98	105
100	9	88	105	9	91	105	9	99	105
101	9	100	105	9	100	105	9	100	105
102	9	101	105	9	101	105	9	101	105
103	9	102	105	9	102	105	9	102	105
104	9	103	105	9	103	105	9	103	105
105	9	103	106	9	103	106	9	103	106
106	9	103	107	9	103	107	9	103	107
107	9	103	120	9	103	120	9	103	120
108	0	0	0	0	0	0	0	0	0

position	Table 1			Table 2			Table 3		
	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed	data rate	ACK seen	ACK missed
109	0	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	6	107	121
111	0	0	0	0	0	0	6	110	121
112	0	0	0	0	0	0	6	111	121
113	0	0	0	0	0	0	6	112	121
114	0	0	0	0	0	0	6	113	121
115	0	0	0	0	0	0	6	114	121
116	6	104	121	6	107	121	6	115	121
117	6	116	121	6	116	121	6	116	121
118	6	117	121	6	117	121	6	117	121
119	6	118	121	6	118	121	6	118	121
120	6	119	121	6	119	121	6	119	121
121	6	119	122	6	119	122	6	119	122
122	6	119	123	6	119	123	6	119	123
123	6	119	120	6	119	120	6	119	120
124	0	0	0	0	0	0	0	0	0
125	0	0	0	0	0	0	0	0	0
126	0	0	0	0	0	0	0	0	0
127	0	0	0	0	0	0	0	0	0

Appendix B

Wait for Response State Code

The following code is from the exit executives of the "Wait for Response"-state located in the OPNET process model for the IEEE 802.11a MAC as described in section 3.1.3. This code checks the Auto Rate Fallback table when an acknowledgement is missed, and might adapt the data rate. The functions starting with `op_` are OPNET specific commands, the so-called "kernel procedures".

```
/* Clear the frame timeout interrupt once the receiver is busy */
/* or the frame is received (in case of collisions, the */
/* frames whose reception has started while we were */
/* transmitting are excluded in the FRAME_RCVD macro). */
intrpt_type = op_intrpt_type ();
if (((intrpt_type == OPC_INTRPT_STAT && op_intrpt_stat () <
          TRANSMITTER_BUSY_INSTAT &&
          op_stat_local_read (op_intrpt_stat ()) == 1.0 &&
          rcv_channel_status == 0) ||
          FRAME_RCVD) &&
          (op_ev_valid (frame_timeout_evh) == OPC_TRUE))
        {
        op_ev_cancel (frame_timeout_evh);
        }

/* Call the interrupt processing routine for each interrupt */
/* request. */

wlan_interrupts_process ();

/* If expected frame is not received in the set */
/* duration or the there is a collision at the */
/* receiver then set the expected frame type to */
/* be none because the station needs to retransmit */
/* the frame. */

/* AJ: Auto Rate Fallback algorithm, missed ACK */

if (FRAME_TIMEOUT)
           {
                if (expected_frame_type == WlanC_Ack)
                        {
```

```
                    current_status = auto_rate_table[current_status][2];
                    op_stat_write (current_status_handle,
                                   (double)current_status);
                    operational_speed =
                            auto_rate_table[current_status][0]*1000000;
                    ack_missed = 1;
                    op_stat_write (ack_missed_handle, (double)ack_missed);
                    }

/* Setting expected frame type to none frame */
expected_frame_type = WlanC_None;

/* retransmission counter will be incremented */
retry_count = retry_count + 1;

/* Reset the NAV duration so that the */
/* retransmission is not unnecessarily delayed. */
nav_duration = current_time;

/* Check whether further retries are possible or */
/* the data frame needs to be discarded. */
wlan_frame_discard ();
}
```

Appendix C

Example Function Code

The code in this appendix is an example of one of the functions defined in the "Function Block" of the OPNET process model for the IEEE 802.11a MAC as described in section 3.1.3. We chose for this specific function to be printed, because in this function the Auto Rate Fallback table is consulted when an acknowledgement arrives.

```
static void
wlan_physical_layer_data_arrival ()
        {
        char   msg_string [120];
        int    dest_addr, src_addr;
        int    accept;
        int    data_pkt_id;
        int    final_dest_addr;
        int    rcvd_sta_bssid;
        WlanT_Data_Header_Fields*     pk_dhstruct_ptr;
        WlanT_Control_Header_Fields*  pk_chstruct_ptr;
        WlanT_Mac_Frame_Type          rcvd_frame_type;
        Packet*                       wlan_rcvd_frame_ptr;
        Packet*                       seg_pkptr;

        /*   802.11a Model Addition   */
        /* Add new variables for the received data rate and the MPDU size of the */
        /* received data packet. */
        /* Adapted from the Philips Labs 802.11a model code (dated 11/15/00). */
        double received_data_rate;
        int    MPDU_size;

        /** Process the frame received from the lower layer.         **/
        /** This routine decapsulate the frame and set appropriate   **/
        /** flags if the station needs to generate a response to the **/
        /** received frame.**/
        FIN (wlan_physical_layer_data_arrival ());

        /*  Access received packet from the physical layer stream. */
        wlan_rcvd_frame_ptr = op_pk_get (i_strm);

        op_pk_nfd_access (wlan_rcvd_frame_ptr, "Accept", &accept);
```

```
/* If the packet is received while the station is in transmission */
/* the packet will not be processed and if needed the station will */
/* need to retransmit the packet.*/
if ((wlan_flags->rcvd_bad_packet == OPC_BOOLINT_ENABLED)
    || (accept == OPC_FALSE))
        {
        /* If the pipeline stage set the accept */
        /* flag to be false then it means that */
        /* the packet is erroneous.  Enable the */
        /* EIFS duration flag and set */
        /* nav duration to be EIFS duration. */
        if (accept == OPC_FALSE)
                {
                wlan_flags->wait_eifs_dur = OPC_BOOLINT_ENABLED;

                /* Setting nav duration to EIFS. */
                nav_duration = current_time + eifs_time;

                /* Reporting the amount of time the channel will be busy. */
                op_stat_write (channel_reserv_handle,
                               (nav_duration - current_time));
                op_stat_write (channel_reserv_handle, 0.0);
                }

        /* We have experienced a collision during transmission. We */
        /* could be transmitting a packet which requires a response */
        /* (a data frame requiring an Ack). Even, this is the */
        /* case, we do not take any action right now and wait for the */
        /* related timers to expire; then we will retransmit the frame. */
        /* This is the approach described in the standard, and it is */
        /* necessary because of the slight possibility that our peer */
        /* may receive the frame without collision and send us the */
        /* response back, which we should be still expecting. */

        /* Check whether the timer for the expected response has */
        /* already expired. If yes, we must initiate the retransmission.*/
        if ((expected_frame_type != WlanC_None)
            && (wlan_flags->transmitter_busy == OPC_BOOLINT_DISABLED)
            && (op_ev_valid (frame_timeout_evh) == OPC_FALSE))
                {
                retry_count = retry_count + 1;

                /* If Rts sent flag was enable then disable it /*
                /* as the station will recontend for the channel. */
                if (wlan_flags->rts_sent == OPC_BOOLINT_ENABLED)
                        {
                        wlan_flags->rts_sent = OPC_BOOLINT_DISABLED;
                        }

                /* Check whether further retries are possible or */
                /* the data frame needs to be discarded. */
                wlan_frame_discard ();
```

```
                /* Set expected frame type flag to none as the station */
                /* needs to retransmit the frame. */
                expected_frame_type = WlanC_None;

                /* Reset the NAV duration so that the */
                /* retransmission is not unnecessarily delayed. */
                nav_duration = current_time;
                }

        /* No frame response will be generated for bad frame. */
        fresp_to_send = WlanC_None;

        /* Reset the bad packet receive flag for subsequent receptions. */
        wlan_flags->rcvd_bad_packet = OPC_BOOLINT_DISABLED;

        /* Printing out information to ODB. */
        if (wlan_trace_active == OPC_TRUE)
                {
                sprintf (msg_string,
                   "Received bad packet. Discarding received packet");
                op_prg_odb_print_major (msg_string, OPC_NIL);
                }

        /* Destroy the bad packet. */
        op_pk_destroy (wlan_rcvd_frame_ptr);

        /* Break the routine as no further processing is needed. */
        FOUT;
        }

/* If waiting for EIFS duration then set the nav duration such that */
/* the normal operation is resumed. */
if (wlan_flags->wait_eifs_dur == OPC_BOOLINT_ENABLED)
        {
        nav_duration = current_time;
        wlan_flags->wait_eifs_dur = OPC_BOOLINT_DISABLED;
        }

/* Getting frame control field and duration information from */
/* the received packet. */
op_pk_nfd_access (wlan_rcvd_frame_ptr, "Type", &rcvd_frame_type) ;

/* Divide processing based on frame type */
switch (rcvd_frame_type)
        {
        case WlanC_Data:
                {
                /** First check that wether the station is expecting **/
                /** any frame or not. If not then decapsulate relevant **/
                /** information from the packet fields and set the **/
                /** frame response variable with appropriate **/
                /** frame type. **/
```

```
/*   802.11a Model Addition   */
/* Extract the size of the MPDU from the */
/* received data frame and report it. */
/* Adapted from the Philips Labs 802.11a */
/* model code (dated 11/15/00). */
op_pk_nfd_access (wlan_rcvd_frame_ptr,
                  "MPDU size", &MPDU_size);
op_stat_write (data_traffic_rcvd_handle_inbits, MPDU_size);
op_stat_write (data_traffic_rcvd_handle_inbits, 0.0);

/*  Data traffic received report */
/* in terms of number of packets. */
op_stat_write (data_traffic_rcvd_handle, 1.0);
op_stat_write (data_traffic_rcvd_handle, 0.0);

/* Address information, sequence control fields, */
/* and the data is extracted from the rcvd packet. */
op_pk_nfd_access (wlan_rcvd_frame_ptr, "Wlan Header",
                  &pk_dhstruct_ptr);

/* Data packet id of the received data frame is extracted. */
op_pk_nfd_access (wlan_rcvd_frame_ptr, "Data Packet ID",
                  &data_pkt_id);

dest_addr = pk_dhstruct_ptr->address1;
remote_sta_addr = pk_dhstruct_ptr->address2;

/* If the station is an AP then it will need */
 /* to forward the receiving data to this address. */
/* Otherwise this field will be zero and will be ignored. */
final_dest_addr = pk_dhstruct_ptr->address3;

fresp_to_send = WlanC_None;

/* Process frame only if it is destined for this station. */
/* Or it is a broadcast frame. */
if ((dest_addr == my_address) || (dest_addr < 0))
        {
        /* Extracting the MSDU from the packet only */
        /* if the packet is destined for this station. */
        op_pk_nfd_get (wlan_rcvd_frame_ptr,
                       "Frame Body", &seg_pkptr);

        /* Only send acknowledgement if the data frame */
        /* is destined for this station. */
        /* No Acks for broadcast frame. */
        if (dest_addr == my_address)
                {
                /* Send the acknowledgement */
                /* to any received data frame. */
                fresp_to_send = WlanC_Ack;
                }
```

```
                /* If its a duplicate packet then destroy */
                /* it and do nothing otherwise, */
                /* insert it in the defragmentation list. */
                if (wlan_tuple_find (remote_sta_addr,
                    pk_dhstruct_ptr->sequence_number,
                    pk_dhstruct_ptr->fragment_number) == OPC_FALSE)
                        {
                        wlan_data_process (seg_pkptr,
                                           remote_sta_addr,
                                           final_dest_addr,
                                           pk_dhstruct_ptr->
                                                fragment_number,
                                           pk_dhstruct_ptr->
                                                more_frag,
                                           data_pkt_id,
                                           rcvd_sta_bssid);
                        }
            }
        else
                {
                /* Printing out information to ODB. */
                if (wlan_trace_active == OPC_TRUE)
                        {
                        sprintf (msg_string,
                                "Data packet %d is
                                received and discarded",
                                data_pkt_id);
                        op_prg_odb_print_major (msg_string,
                                                OPC_NIL);
                        }

                /* If the frame is not destined for this station */
                /* then do not respond with any frame. */
                fresp_to_send = WlanC_None;
                }

        if (expected_frame_type != WlanC_None)
                {
                /* Since the station did not */
                /* receive the expected frame */
                /* it has to retransmit the packet. */
                retry_count = retry_count + 1;

                /* If Rts sent flag was enable then disable it as */
                /* the station will recontend for the channel.*/
                if (wlan_flags->rts_sent == OPC_BOOLINT_ENABLED)
                        {
                        wlan_flags->rts_sent = OPC_BOOLINT_DISABLED;
                        }

                /* Reset the NAV duration so that the */
                /* retransmission is not unnecessarily delayed. */
                nav_duration = current_time;
```

```
                        }

                /* Update nav duration if the received nav */
                /* duration is greater */
                /* than the current nav duration. */
                if (nav_duration < (pk_dhstruct_ptr->
                                        duration + current_time))
                        {
                        nav_duration = pk_dhstruct_ptr->
                                                duration + current_time;

                        /* Set the flag that indicates updated NAV value. */
                        wlan_flags->nav_updated = OPC_BOOLINT_ENABLED;
                        }
                break;
                }

        case WlanC_Ack:
                {
                /* AJ:  Auto Rate Fallback Algorithm, seen ACK.*/
                if (expected_frame_type == WlanC_Ack)
                        {
                        current_status = auto_rate_table[current_status][1];
                           op_stat_write (current_status_handle,
                                          (double)current_status);
                        operational_speed =
                                auto_rate_table[current_status][0]*1000000;
                        ack_missed = 0;
                        op_stat_write (ack_missed_handle, (double)ack_missed);
                        }

                /* No response needed for ack frame. */
                fresp_to_send = WlanC_None;

                op_pk_nfd_access (wlan_rcvd_frame_ptr,
                                "Wlan Header", &pk_chstruct_ptr);
                dest_addr = pk_chstruct_ptr->rx_addr;

                /* AJ: Count number of ACKs received */
                ack_rcvd = ack_rcvd + 1;
                op_stat_write (ack_rcvd_handle, (double)ack_rcvd);

                /*  Control Traffic received report */
                /* in terms of number of bits. */
                op_stat_write (ctrl_traffic_rcvd_handle_inbits,
                               (double) WLAN_ACK_LENGTH);
                op_stat_write (ctrl_traffic_rcvd_handle_inbits, 0.0);

                /*  Control Traffic received report */
                /*in terms of number of packets.*/
                op_stat_write (ctrl_traffic_rcvd_handle, 1.0);
                op_stat_write (ctrl_traffic_rcvd_handle, 0.0);
```

```
if ((dest_addr == my_address)
    && (rcvd_frame_type == expected_frame_type))
        {
        /* Printing out information to ODB. */
        if (wlan_trace_active == OPC_TRUE)
                {
                sprintf (msg_string,
                        "Ack received for data packet %d",
                        pkt_in_service);
                op_prg_odb_print_major (msg_string, OPC_NIL);
                }

        op_stat_write (retrans_handle,
                       (double) (retry_count * 1.0));
        op_stat_write (retrans_handle, 0.0);

        /* Reset the retry counter as */
        /* the expected frame is received */
        retry_count = 0;

        /* Decrement number of fragment count */
        /* because one fragment is successfully transmitted. */
        num_fragments = num_fragments - 1;

        /* When there are no more fragments to transmit then */
        /* disable the Rts sent flag */
        /* if it was enabled because the contention */
        /* period due to Rts/Cts exchange is */
        /* over and another Rts/Cts exchange is */
        /* needed for next contention period. */
        if ((num_fragments == 0)
                && (wlan_flags->rts_sent
                    == OPC_BOOLINT_ENABLED))
                {
                wlan_flags->rts_sent
                        = OPC_BOOLINT_DISABLED;

                /* Set the contention window flag. */
                /* Since the ACK for the last */
                /* fragment indicates a successful */
                /* transmission of the entire data, */
                /* we need to back-off for a */
                /* contention window period. */
                wlan_flags->cw_required = OPC_TRUE;
                }

        /* Data packet is successfully */
        /* delivered to remote station, */
        /* since no further retransmission */
        /* is needed the copy of the data */
        /* packet will be destroyed. */
        op_pk_destroy (wlan_transmit_frame_copy_ptr);
        wlan_transmit_frame_copy_ptr = OPC_NIL;
```

```
                }

        else
                {
                /* Printing out information to ODB. */
                if (wlan_trace_active == OPC_TRUE)
                        {
                        sprintf (msg_string,
                                "Ack is received and discarded.");
                        op_prg_odb_print_major (msg_string, OPC_NIL);
                        }

                /* Check whether we were expecting */
                /* another frame. If yes then */
                /* we need to retransmit the frame */
                /* for which we were expecting */
                /* a reply. */
                if (expected_frame_type != WlanC_None)
                        {
                        /* Since the station did not */
                        /* receive the expected frame */
                        /* it has to retransmit the packet */
                        retry_count = retry_count + 1;

                        /* If Rts sent flag was enable */
                        /* then disable it as the station */
                        /* the station will recontend */
                        /* for the channel. */
                        if (wlan_flags->rts_sent
                                        == OPC_BOOLINT_ENABLED)
                                {
                                wlan_flags->rts_sent
                                        = OPC_BOOLINT_DISABLED;
                                }

                        /* Reset the NAV duration so that the */
                        /* retransmission is not unnecessarily
                        /* delayed. */
                        nav_duration = current_time;
                        }
                }

        /* If network allocation vector is less than /*
        /* the received duration */
        /* value then update its value. */
        if (nav_duration < (pk_chstruct_ptr->
                            duration + current_time))
                {
                nav_duration = pk_chstruct_ptr->
                               duration + current_time;

                /* Set the flag that */
                /* indicates updated NAV value. */
```

```
                        wlan_flags->nav_updated = OPC_BOOLINT_ENABLED;
                        }
                break;
                }

        default:
                {
                wlan_mac_error ("Unexpected frame type received.",
                                OPC_NIL, OPC_NIL);
                break;
                }
    }

/* Reporting the amount of time the channel will be busy. */
op_stat_write (channel_reserv_handle, (nav_duration - current_time));
op_stat_write (channel_reserv_handle, 0.0);

/* Check whether further retries are possible or */
/* the data frame needs to be discarded. */
wlan_frame_discard ();

/* Set the expected frame type to None because either the */
/* expected frame is recieved or the station will have to */
/* retransmit the frame */
expected_frame_type = WlanC_None;

/* Destroying the received frame once */
/* relevant information is taken out of it.*/
op_pk_destroy (wlan_rcvd_frame_ptr);

FOUT;
}
```

Appendix D

Received Power Model Stage Code

The code of the "received power" stage of the transceiver pipeline as described in section 3.2.1 is printed in this appendix. This particular stage includes the code of the Path Loss-model used.

```
/* wlan_power.ps.c */
/* Default received power model for radio link Transceiver Pipeline */


/* This version of this pipeline stage is designed for */
/* use with the 802.11a Models */
/* created by Bryan Braswell at the Naval Postgraduate School.*/


/***************************************/
/*     Copyright (c) 1993-2000 */
/*      by OPNET Technologies, Inc.     */
/* (A Delaware Corporation)*/
/* 3400 International Drive,  N.W. */
/* Washington, D.C., U.S.A.*/
/* All Rights Reserved.    */
/***************************************/

#include "opnet.h"
#include <math.h>


/***** constants *****/

#define C 3.0E+08 /* speed of light (m/s) */
#define SIXTEEN_PI_SQ 157.91367 /* 16 times pi-squared */

static const char* PowI_Err_Hdr =
  "Error in radio power computation pipeline stage (dra_power):";


/***** pipeline procedure *****/

#if defined (__cplusplus)
```

```
extern "C"
#endif
void
wlan_power (Packet * pkptr)
        {
        double                prop_distance, rcvd_power, path_loss;
        double                tx_power, tx_base_freq, tx_bandwidth, tx_center_freq;
        double                lambda, rx_ant_gain, tx_ant_gain;
        Boolean               sig_lock;
        Objid                rx_ch_obid, rx_obid;
        double                in_band_tx_power, band_max, band_min;
        double                rx_base_freq, rx_bandwidth;
        double                rx_reception_end, pk_reception_end;

        /** Compute the average power in Watts of the **/
        /** signal associated with a transmitted packet. **/
        FIN (dra_power (pkptr));

        /* If the incoming packet is 'valid', it may cause the receiver to */
        /* lock onto it. However, if the receiving node is disabled, then */
        /* the channel match should be set to noise.*/
        if (op_td_get_int (pkptr, OPC_TDA_RA_MATCH_STATUS) == OPC_TDA_RA_MATCH_VALID)
                {
                if (op_td_is_set (pkptr, OPC_TDA_RA_ND_FAIL))
                        {
                        /* The receiving node is disabled.  Change */
                        /* the channel match status to noise. */
                        op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS,
                                       OPC_TDA_RA_MATCH_NOISE);
                        }
                else
                        {
                        /* The receiving node is enabled. Get the object id */
                        /* of the receiver channel and the receiver itself. */
                        rx_ch_obid = op_td_get_int (pkptr, OPC_TDA_RA_RX_CH_OBJID);
                        rx_obid    = op_topo_parent (op_topo_parent (rx_ch_obid));

                        /* Obtain the status of the receiver. If end of */
                        /* reception time is in future then this means the */
                        /* receiver is already busy. */
                        op_ima_obj_attr_get (rx_obid, "reception end time",
                                             &rx_reception_end);

                        /* If the receiver is already receiving another */
                        /* packet, then the packet will now be considered */
                        /* to be noise. This prevents simultaneous */
                        /* reception of multiple valid packets on any given */
                        /* radio channel and the entire radio receiver, */
                        /* since in the wlan nodes, all the channels use */
                        /* the same frequency. */
                        if (rx_reception_end <= op_sim_time ())
                                {
                                /* The receiver is idle. Turn on the signal */
```

```
                        /* lock. No need to change the status of the */
                        /* packet, since it is valid by default. */
                        sig_lock = OPC_BOOLINT_ENABLED;
                        op_ima_obj_attr_set (rx_ch_obid,
                                             "signal lock", sig_lock);
                        }
                else
                        {
                        /* At least one channel of the receiver is */
                        /* busy. We will handle the current packet as */
                        /* noise. */
                        op_td_set_int (pkptr, OPC_TDA_RA_MATCH_STATUS,
                                       OPC_TDA_RA_MATCH_NOISE);
                        }

                /* Update the reception end time for the receiver */
                /* based on the new packet. */
                pk_reception_end = op_td_get_dbl (pkptr,
                                                  OPC_TDA_RA_END_RX);
                if (pk_reception_end > rx_reception_end)
                        op_ima_obj_attr_set (rx_obid,
                                             "reception end time",
                                             pk_reception_end);
                }
        }

/* Get power allotted to transmitter channel. */
tx_power = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_POWER);

/* Get transmission frequency in Hz. */
tx_base_freq = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_FREQ);
tx_bandwidth = op_td_get_dbl (pkptr, OPC_TDA_RA_TX_BW);
tx_center_freq = tx_base_freq + (tx_bandwidth / 2.0);

/* Caclculate wavelength (in meters). */
lambda = C / tx_center_freq;

/* Get distance between transmitter and receiver (in meters). */
prop_distance = op_td_get_dbl (pkptr, OPC_TDA_RA_START_DIST);

/* When using TMM, the TDA OPC_TDA_RA_RCVD_POWER will already */
/* have a raw value for the path loss. */
if (op_td_is_set (pkptr, OPC_TDA_RA_RCVD_POWER))
        {
        path_loss = op_td_get_dbl (pkptr, OPC_TDA_RA_RCVD_POWER);
        }
else
        {
        /* Compute the path loss for this distance and wavelength. */
        if (prop_distance > 0.0)
                {
                /* AJ: New Path Loss model */
                if (prop_distance < 5.0)
```

```
                        {
                        path_loss = (lambda * lambda) /
                        (SIXTEEN_PI_SQ * pow(prop_distance, 2.0));
                        }
                else
                        {
                        path_loss = (lambda * lambda) /
                        (SIXTEEN_PI_SQ * 5 * 5 * pow(prop_distance, 3.3) /
                        pow(5, 3.3));
                        }
                }
        else
                path_loss = 1.0;
        }

/* Determine the receiver bandwidth and base frequency. */
rx_base_freq = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_FREQ);
rx_bandwidth = op_td_get_dbl (pkptr, OPC_TDA_RA_RX_BW);

/* Use these values to determine the band overlap with the transmitter. */
/* Note that if there were no overlap at all, the packet would already */
/* have been filtered by the channel match stage. */

/* The base of the overlap band is the highest base frequency. */
if (rx_base_freq > tx_base_freq)
        band_min = rx_base_freq;
else
        band_min = tx_base_freq;

/* The top of the overlap band is the lowest end frequency. */
if (rx_base_freq + rx_bandwidth > tx_base_freq + tx_bandwidth)
        band_max = tx_base_freq + tx_bandwidth;
else
        band_max = rx_base_freq + rx_bandwidth;

/* Compute the amount of in-band transmitter power. */
in_band_tx_power = tx_power * (band_max - band_min) / tx_bandwidth;

/* Get antenna gains (raw form, not in dB). */
tx_ant_gain = pow (10.0, op_td_get_dbl (pkptr, OPC_TDA_RA_TX_GAIN) / 10.0);
rx_ant_gain = pow (10.0, op_td_get_dbl (pkptr, OPC_TDA_RA_RX_GAIN) / 10.0);

/* Calculate received power level. */
rcvd_power = in_band_tx_power * tx_ant_gain * path_loss * rx_ant_gain;

/* Assign the received power level (in Watts) */
/* to the packet transmission data attribute. */
op_td_set_dbl (pkptr, OPC_TDA_RA_RCVD_POWER, rcvd_power);

FOUT;
}
```