Easing the transition from inspiration to implementation: A rapid prototyping platform for wireless medium access control protocols

A thesis
submitted in partial fulfilment
of the requirements for the degree
of
Doctor of Philosophy
at
The University of Waikato
by
DEAN ANDREW ARMSTRONG

The University of Waikato
2007

Abstract

Packet broadcast networks are in widespread use in modern wireless communication systems. Medium access control is a key functionality within such technologies. A substantial research effort has been and continues to be invested into the study of existing protocols and the development of new and specialised ones. Academic researchers are restricted in their studies by an absence of suitable wireless MAC protocol development methods.

This thesis describes an environment which allows rapid prototyping and evaluation of wireless medium access control protocols. The proposed design flow allows specification of the protocol using the specification and description language (SDL) formal description technique. A tool is presented to convert the SDL protocol description into a C++ model suitable for integration into both simulation and implementation environments.

Simulations at various levels of abstraction are shown to be relevant at different stages of protocol design. Environments based on the Cinderella SDL simulator and the *ns-2* network simulator have been developed which allow early functional verification, along with detailed and accurate performance analysis of protocols under development.

A hardware platform is presented which allows implementation of protocols with flexibility in the hardware/software trade-off. Measurement facilities are integral to the hardware framework, and provide a means for accurate real-world feedback on protocol performance.

Acknowledgements

Numerous people have supported me directly and indirectly in various ways through the duration of the work described herein. Though there are too many to name, some require special mention.

Firstly I would like to express my sincere gratitude to Dr. Murray Pearson whose vision, insight and ability to motivate have been vital factors in the success of the research described within this thesis. Dr. Tony McGregor, as my second supervisor, was always available for discussion and able to provide inspiring comment, and I am grateful to him for this.

I would like to thank the other staff and students of the WAND Network Research Group in the Department of Computer Science at The University of Waikato. I cannot imagine a more enjoyable or supportive environment in which to work. In particular I thank my colleagues and friends Jamie Curtis and James Spooner who were constantly approached as sounding boards for ideas, and always made time to chat. I am also grateful for the efforts of fellow students Scott Raynel, Sam Bartels, and Andreas Lööw whose work using various iterations of the development framework described herein has provided much valuable feedback.

This work has been supported financially by The University of Waikato; the Department of Computer Science; and the Sir Edmund Hillary Scholarship Program, and I am indebted to these organisations.

My parents, Bill and Estelle, have always endeavoured to support me in every possible way that they could. I am eternally grateful for their constant love and support. Finally, I wish to thank my darling Deborah for her love and understanding in the face of ever-extending submission dates.

Contents

List of Figures

List of Terms

ACK Acknowledgement

API Application Programming Interface

AP Access Point

ARP Address Resolution Protocol

ARQ Automatic Repeat Request

CCA Clear Channel Assessment

CDMA Code Division Multiple Access

CRC Cyclic Redundancy Check

CSMA Carrier Sense Multiple Access

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

DCF Distributed Coordination Function

DSP Digital Signal Processor

DSSS Direct Sequence Spread Spectrum

FAMA Floor Acquisition Multiple Access

FDD Frequency Division Duplex

FDMA Frequency Division Multiple Access

FDT Formal Description Technique

FHSS Frequency Hopping Spread Spectrum

FIFO First-in, First-out

FPGA Field Programmable Gate Array

FTP File Transfer Protocol

GPIO General Purpose Input/Output

GPS Global Positioning System

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronic Engineers

IP Internet Protocol

ISM Industrial, Scientific and Medical

ISO International Organization for Standardization

LAN Local Area Network

LLC Logical Link Control

LOTOS Language of Temporal Ordering Specification

MAC Medium Access Control

MADWIFI Multi-band Atheros Driver for Wi-Fi

MAN Metropolitan Area Network

MIMO Multiple Input, Multiple Output

MLME MAC Sublayer Management Entity

MPDU MAC Protocol Data Unit

MSC Message Sequence Chart

MSDU MAC Service Data Unit

NAV Network Allocation Vector

NIC Network Interface Card

OSI Open Systems Interconnect

OTcl Object-oriented Tool Command Language

PAN Personal Area Network

PCI Peripheral Component Interconnect

PDA Personal Digital Assistant

PDU Protocol Data Unit

PHY Physical Layer

PLCP Physical Layer Convergence Protocol

PSDU Physical layer Service Data Unit

PSK Phase Shift Keying

QoS Quality of Service

RAM Random Access Memory

RF Radio Frequency

RSSI Received Signal Strength Indication

RTP Real-time Transport Protocol

SDL Specification and Description Language

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

TCP Transmission Control Protocol

TDD Time Division Duplex

TDMA Time Division Multiple Access

UDP User Datagram Protocol

UWB Ultra-Wide Band

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

VoIP Voice over Internet Protocol

WAG Wireless Analysis and Generation

WLAN Wireless Local Area Network

WPAN Wireless Personal Area Network

WPDE Waikato Protocol Development Environment

Chapter 1

Introduction

The field of Medium Access Control (MAC) protocol design for wireless computer networks is one which has seen dramatic advances in the state of knowledge in its short history. From their beginnings in the ALOHA system [1], wireless MAC protocols have progressed from simple access control schemes to complex entities incorporating encryption, forward error correction, and other facilities.

Wireless Local Area Network (WLAN) and Wireless Personal Area Network (WPAN) technologies have been areas of significant development. Examples of these such as IEEE 802.11 [2] (promoted by the Wi-Fi Alliance [3]) and Bluetooth [4] are ubiquitous, and continually developed to meet the needs of changing applications and underlying radio technology.

Another area of ongoing development is the application of wireless technologies to high-bandwidth infrastructure links. Technologies such as IEEE 802.16 [5] (including profiles promoted by the WiMAX Forum [6]) and HiperMAN [7] define MAC protocols optimised for providing broadband 'last-mile' connectivity. Such protocols require the ability to guarantee a level of service to a subscriber to allow, for example, reliable Voice over Internet Protocol (VoIP) services.

Cellular systems have been the major driving force for MAC protocol devel-

opment in recent years. The ever-increasing bandwidth needs of multimedia handsets have presented challenges for protocol designers and pushed the development of power efficient, high performance MAC protocols. Extensions to the IEEE 802.16 standard provide for mobile nodes and low power operation, allowing it to be targeted at the cellular market in the form of Mobile WiMAX [6] and WiBro [8]. The UMTS standard [9] is specifically targeted at this application, and with high-throughput extensions such as HSDPA [10] provides broadband access to the handset.

Handset design has brought about another significant issue for the current generation of network protocols – the impending convergence of WLAN, WPAN and Cellular technologies. As products are developed with higher levels of integration, interference between collocated wireless technologies which share the same or similar bands of operation will require MAC protocols which are able to coexist with others to efficiently share the available spectrum [11, 12].

Advances in wireless physical layer technologies are a major driving force for MAC protocol development. Variants of the IEEE 802.11 protocol have evolved from the original one megabit per second (Mbps) raw throughput to the coming IEEE 802.11n which provides 540Mbps. Such progress appears set to continue across the field with Ultra-Wide Band (UWB) technologies attracting a large amount of research effort currently.

With developments at the physical layer, new MAC protocols are required to make efficient use of the bandwidth available. Newly conceived applications for these technologies, and the ever-increasing demands that they place on the MAC protocols, also requires constant effort in this area. These factors ensure that MAC protocol development is currently relevant and will continue to be well into the future. As such, it is important that design methodologies for these systems are well understood.

1.1 MAC Design Process

MAC protocols have unique requirements due to their position in the network stack. By definition they must interact closely with the physical layer, meaning strict observance of often-tight timing requirements. A conflicting demand is brought by increased pressure for low-level functionality to support facilities such as Quality of Service (QoS) and security.

Modern MAC protocols are complex distributed real-time algorithms, which are often implemented in embedded systems. They must be reliable and fault-tolerant. It is generally accepted that systems of this nature benefit greatly from use of formal methods in the specification stage [13, 14, 15]. However, formal specification in the design stages is not sufficient to ensure a successful protocol. Typically various representations are required at different stages of the design life-cycle. Manual translation between these representations leaves the process susceptible to human error. Automatic translation alleviates this problem, but refinement of different representations can lead to inconsistencies between the various forms. The ideal design flow will allow refinement of an initial specification through to prototype implementation.

Wireless protocol research is an active field in academia, however a lack of suitable development frameworks means that the design process is typically disjointed. Most studies are based around analytical or simulative techniques, which provide a useful and accessible means of evaluating protocols, but require care in their use as the factors which may affect the validity of their results are generally not well understood [16, 17, 18]. Prototype implementation can enable more thorough evaluation, however a lack of suitable implementation platforms has kept this approach beyond the reach of the majority of academic researchers. The development of a consistent design flow from initial specification through to implementation would encourage continual development in this area with use of sound design practices.

Figure 1.1 illustrates the idealised design flow for MAC protocols which has been developed within the context of this thesis. The flow consists of an iterative refinement of both the protocol and an evaluation testbench. This

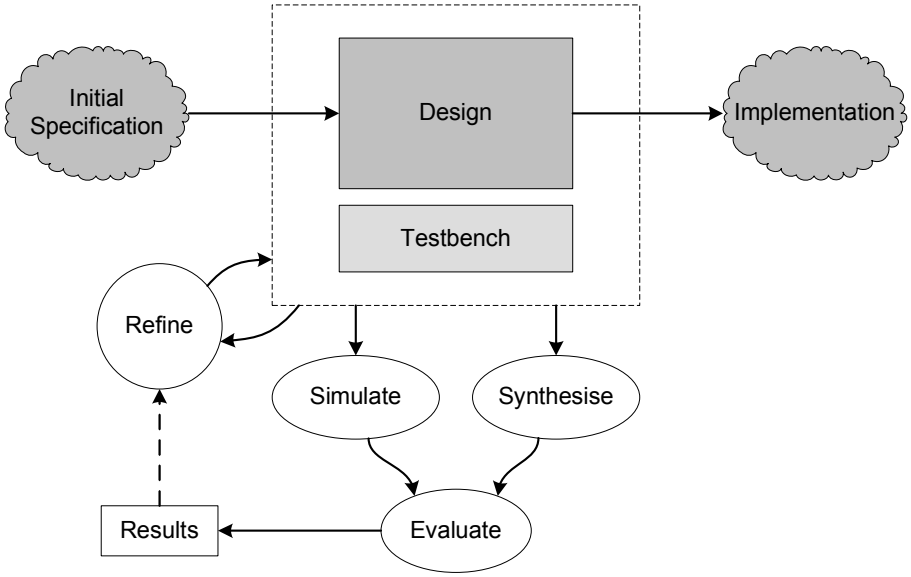


Figure 1.1: MAC Protocol Design Cycle

refinement is driven by the results of evaluation of the protocol under development against the testbench, and the addition of detail in a progression towards a final implementation.

Evaluation at any given stage may comprise simulation, emulation, or synthesis to and measurement of an implementation. Different aspects of protocol operation are targets for evaluation at different stages of the design life-cycle. For this reason the evaluation testbench must evolve along with the protocol over the design cycle.

A top-down design methodology is key to this design flow. The top-down paradigm promotes architectural planning and early emphasis on the interactions between logical blocks – important elements in the design of a robust protocol. A disadvantage of the traditional top-down method is the inability to test the design at early stages of the process. Automated translation of early specifications into simulation models and implementations goes some way to alleviating this limitation.

1.2 Statement of Problem

Environments which support a consistent design methodology for wireless Medium Access Control protocols are not readily available to researchers in this field. An ideal platform would allow seamless progression from an initial specification through to a prototype implementation, incorporating evaluation at all stages of the design lifetime using techniques such as simulation with appropriate models, and measurement of implementation performance. Ready access to such a platform would foster research and therefore the accelerated advancement of knowledge in this area.

The objective set at the outset of the work described in this document was to develop a framework supporting the idealised design flow for wireless MAC protocols shown in Figure 1.1, and demonstrate the effectiveness of such an approach.

As such it is the statement of this thesis that: *Rapid development of MAC protocols requires a common and consistent evaluation framework. A rapid prototyping method for MAC protocols incorporating such a framework is both useful and feasible.*

1.3 Thesis Outline

This work describes the design of the Waikato Protocol Development Environment (WPDE) – an environment for the specification, development, simulation, implementation and evaluation of wireless media access control protocols. A seamless design flow allows the use of a common representation which is refined from specification through to a prototype implementation.

The structure of this thesis is as follows:

In Chapter 2 we discuss the various roles and responsibilities of modern wireless medium access control protocols in an effort to understand the range of

functionality which must be captured within representations of such protocols. Recent work in the area of MAC protocol design and evaluation is summarised, and several relevant protocols are presented as the state-of-the-art in this field.

Chapter 3 discusses the benefits of formal methods in the development of MAC protocols. The LOTOS, ESTELLE, and SDL languages are introduced and compared in terms of their ability to capture the required functionality, and the ease of their use. The Specification and Description Language (SDL) is presented as the Formal Description Technique (FDT) of choice for the framework presented herein.

Chapter 4 builds toward the definition of generic interfaces for MAC protocols within the development framework. A set of service primitives are defined which provide for the functionality typically required by modern MAC protocols. We define a mapping strategy from an SDL representation of a MAC protocol onto C++ source code to allow compilation into target simulation and implementation environments. Finally, a tool is described which implements the SDL-to-C++ mapping, allowing automatic translation from the initial specification.

Chapter 5 discusses the evaluation of MAC protocols through simulation. The advantages and disadvantages of simulation at various levels of abstraction are discussed, and a system allowing simulation of MAC protocols implemented within the framework described in Chapter 4 is presented. This environment allows simulation at varying levels of detail, and supports realistic traffic and channel models.

Chapter 6 describes a platform which allows implementation of developed protocols. The environment consists of custom hardware, embedded software, host drivers and tools. A modular design allows rapid and automated implementation of protocols to provide for effective evaluation with real-world applications.

Chapter 7 discusses the evaluation of protocols through measurement. It presents a case for the integration of measurement capabilities into the hardware platform, but discusses the implementation challenges that this presents.

It also discusses the need for achieving a large degree of accuracy in timing and the hardware support necessary for this to happen. A hardware and software framework is described which allows in-system measurement of protocols operating within the implementation platform specified in Chapter 6.

Finally, Chapter 8 summarises and draws conclusions from this work.

Chapter 2

Background

The field of wireless computer networking has progressed a great deal since its origins in the ALOHA system [1] developed at the University of Hawaii in the late 1960s. The ALOHA network used two 24 kilo-baud Ultra-High Frequency (UHF) radio channels to provide connectivity between a central node and outlying stations. The shared up-link provided the first example of a packet broadcast channel for computer communication, and the techniques applied by Abramson et al. laid the foundation for such successful protocols as Ethernet (IEEE 802.3) and Wi-Fi (IEEE 802.11).

Packet broadcast systems such as these provide a good solution for networks carrying bursty traffic – that is traffic which has a high ratio of peak to average data rates [19]. Such systems take advantage of the law of large numbers, which infers that the demand for the shared medium at a given instant will likely be similar to the average demand across all stations participating in the network. Clearly there are instants when demand will exceed resource availability, and it is the key role of the Medium Access Control (MAC) to facilitate fair and efficient use of the medium in this case. This functionality is typically considered to be encapsulated by the MAC sub-layer within the common Open Systems Interconnect (OSI) layered reference model for network systems defined by the International Organization for Standardization (ISO). The relationship between this model and the Transmission Control Protocol (TCP),

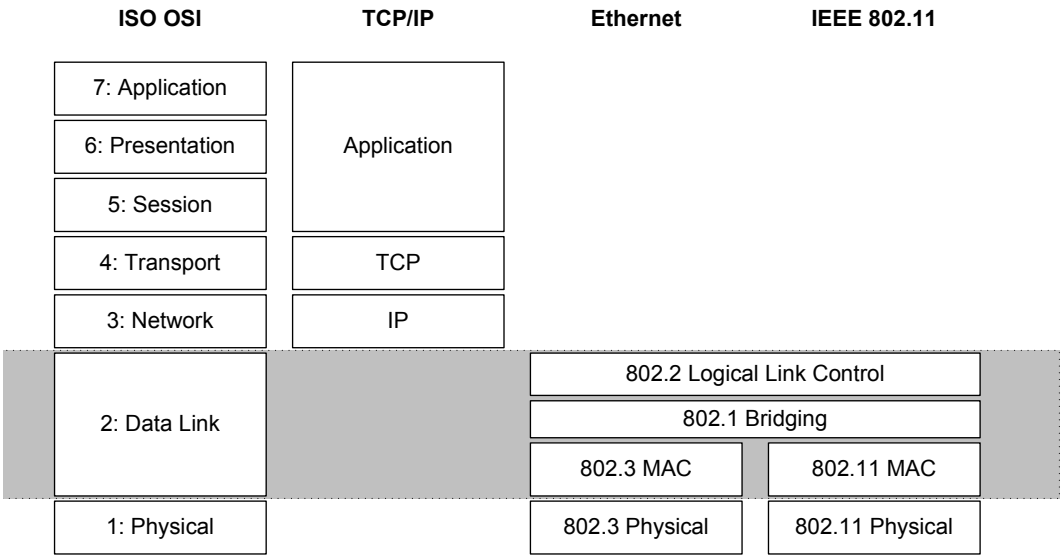


Figure 2.1: OSI, TCP/IP, Ethernet and 802.11 Layered Models

Internet Protocol (IP), Ethernet, and IEEE 802.11 systems is illustrated in Figure 2.1.

The role of the MAC sub-layer in controlling the physical layer brings interactions which break the traditional layered model for network systems. MAC techniques, such as Carrier Sense Multiple Access (CSMA), Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA) and Frequency Division Multiple Access (FDMA), all have distinct requirements in terms of physical layer parameters which must be controlled on a real-time basis. Rapid advances in technology at the physical layer drive the development of new protocols to efficiently make use of the raw bandwidth available.

Concurrently, higher layers of the network stack are forcing more complexity into the MAC sub-layer. Trends toward network applications with higher bandwidth needs, and specific service parameter requirements, coupled with the performance mismatch in recent years between state-of-the-art wired and wireless technologies has led to the application of QoS techniques to wireless networks in an effort to maximise user experience for interactive applications. Effective provision of QoS requires support at the data-link layer and specifically within the MAC sub-layer. Similarly, security features such as authen-

tication and privacy also require mechanisms at this level. Although its key role is to share the medium between participating stations, the modern MAC protocol is increasingly taking responsibility for a larger set of diverse and complex tasks.

Given the complexities of the MAC sub-layer and its integral role in the network stack, it is no surprise that much research effort is spent in this field. Various approaches to practical protocol research are possible, but all are centred around the evaluation of some aspect of protocol operation through one or more of simulation, emulation, or measurement of an implementation. Simulation provides the basis of most studies with its low cost and inherent flexibility, however an appropriate level of detail in the simulation models is required to achieve valid results in a timely fashion. Insufficient detail in simulation models can result in inaccurate or misleading results. Unfortunately, an increase in detail generally leads to an increase in simulation run-time and resource utilisation. In particular it is necessary to use appropriate transport layer and traffic models for the intended application when evaluating lower layers of the network stack.

A number of simulators have been developed for network research. OMNeT++ [20] has a powerful simulation engine coupled with a good graphical interface making it appropriate for use in education, academic research and industry. GloMoSim [21] is an environment designed to enable detailed simulation of large scale communication networks through use of parallel execution techniques. GTNetS [22] further optimises simulation for very large scale topologies. The most common simulator used in network research [22, 23], however, is the *ns-2* simulator [24, 25, 26].

The open-source *ns-2* simulator uses a combination of Object-oriented Tool Command Language (OTcl) and C++ to provide ready reconfiguration of the simulated network, and high performance for processor intensive aspects respectively. It incorporates models at varying levels of abstraction, allowing both high-performance abstract simulations as well as more detailed but correspondingly processor intensive ones. Realistic transport protocol models are provided including numerous variants of TCP, and the User Datagram

Protocol (UDP). Application models include constant and variable bit-rate traffic generators, Telnet, File Transfer Protocol (FTP), and Hypertext Transfer Protocol (HTTP) Server, Client and Caches. Various queue management algorithms are supported, and packets can be marked with QoS information allowing evaluation of QoS-aware protocols. Broch et al. [27] have extended *ns-2* to include implementations of various multi-hop wireless ad hoc routing protocols. The same work has provided support for simulation of wireless nodes including node mobility, and wireless physical layer characteristics like propagation delay, signal capture, and carrier sense. These features are tightly coupled with an IEEE 802.11 Distributed Coordination Function (DCF) protocol model.

Simulation techniques have been widely applied for evaluation both of existing protocols and proposed extensions. Subramanian et al. [28] use *ns-2* to investigate the effect of 802.11b transmission errors caused by an active interferer on TCP performance. They propose reconsideration of some aspects of 802.11 rate-adaptation approaches, and modifications to the TCP algorithm to improve performance in such links. Holland et al. [29] modify *ns-2* with enhanced channel models to support their investigation into rate adaptation techniques which culminates in the proposal of a receiver-based rate adaptation extension to the 802.11 protocol. Fullmer and Garcia-Luna-Aceves [30] use simulation to analyse the performance of Floor Acquisition Multiple Access (FAMA) protocols in networks containing hidden nodes. Korakis et al. [31] propose a MAC protocol for use with steerable directional antenna. They evaluate their system using a custom simulation environment which provides simplistic channel and traffic models.

After simulation in the design lifetime of a process, emulation typically provides the next level of accuracy by allowing the introduction of real network traffic into a simulated network subsystem. This approach maintains the flexibility of simulation, but may require significant processing resources. The *ns-2* simulator supports network emulation through a special simulation scheduler which tracks real-time, and simulation objects which allow capture and generation of live network traffic. A Berkeley Software Distribution (BSD) Packet Filter [32] interface allows selective capture of live network traffic and insertion

into the simulation. Similarly, traffic can be injected into the live network from a special simulation entity which makes use of the UNIX raw socket interface.

Emulation only provides an advantage over a corresponding simulation in the case where the simulation traffic models are lacking in some aspect of detail which has effect on the investigation. This advantage aside, using emulation may impose restrictions on the size of the network to be emulated due to the hardware requirements; any chance of the process completing in less than real-time is also lost; and the processor overhead associated with emulation may affect the validity of the model. Though few studies utilising emulation are evident in the literature, it is still regarded as a useful tool for validation of simulation traffic models and for use in situations where appropriate traffic models do not exist.

Understanding of the relevance of detail in simulation and emulation models comes about through measurement of protocol implementations in real-world scenarios [18], and by definition this is also the most accurate form of evaluation for a protocol under development. By providing for implementation and measurement of a protocol under development in a representative environment the researcher can have a higher degree of confidence in the validity of evaluation results. Despite these reasons, and the known limitations of existing wireless simulation models [33], limited use of measurement is apparent in the literature.

Several studies have focused on link errors in 802.11 networks. Researchers in the MIT Roofnet project investigated packet loss rates within their testbed 802.11b network [34]. Their experiment involved transmitting 802.11 broadcast packets from one node on an idle network, and recording the received packets, along with their signal-strength indications as reported by the network card at destination nodes. Packet transmission and reception times were also recorded, and the data was used to draw conclusions on the characteristics of the wireless link. A drawback of this approach is that packet loss and corruption are indistinguishable. Further, no information on the nature of bit errors within a packet was recorded. A similar methodology is used in work described by Chebrolu et al. [35], with a modified driver providing

packet reception parameters, including detail on success or failure of the Cyclic Redundancy Check (CRC) check. This work provides insight into the error characteristics of long distance 802.11b links.

Similarly, Gross and Willig [36] performed more detailed measurements of 802.11b link error characteristics using a MAC-less radio module. They used two nodes – one transmitting a known series of packets, and the other capturing all received data. Comparison of the original and received packet data provided an indication of transmission errors, from which conclusions were drawn. The same technique was applied in a different Radio Frequency (RF) environment in [37].

The limited use of implementation and measurement based studies in wireless protocol research can be explained by the scarcity of suitable development platforms. Though much research has focused on MAC protocol development, relatively few attempts have been made to provide environments supporting such research. Ganz et al. [38] propose a development environment whereby an off-the-shelf 802.11 Network Interface Card (NIC) is loaded with custom firmware, and the MAC is implemented on the host at the driver level. The MAC developer utilises an Application Programming Interface (API) which provides basic timer functionality and the ability to transmit and receive packets. A similar approach uses standard 802.11 hardware and implements MAC functionality in the device driver with each Physical layer Service Data Unit (PSDU) carried in an 802.11 broadcast packet. This approach is common in the literature [39, 40], and is further enabled by the Multi-band Atheros Driver for Wi-Fi (MADWIFI) [41]. The MADWIFI project provides an open-source driver which allows a large degree of control in off-the-shelf Atheros based wireless NICs. Several projects have used this as a platform for wireless MAC protocol research [42, 43, 44, 45, 46].

These approaches lead to ease of implementation, but latency across the host interface restricts the ability to meet the tight timing constraints generally involved in MAC protocols. Firmware implementation is generally not possible for the academic researcher due to commercial sensitivity around wireless network hardware, and the resultant reluctance of device manufacturers to release

firmware APIs and embedded development tool-chains. Further, in commercial designs fixed hardware often implements key protocol features – meaning that these are likely to be inflexible even if the API were to be available.

The TUTWLAN project [47] has developed a wireless network architecture for use in providing connectivity for multimedia applications between a base station and multiple portable terminals. A medium access control protocol known as TUTMAC provides TDMA to the channel. A large focus of this work has been the development of QoS facilities and mechanisms for the wireless network, however it is the underlying development platform that has particular relevance to this thesis.

The design flow used for the TUTMAC protocol consists of protocol specification in SDL, and translation – using the commercial Telelogic TAU suite – to C source code for porting to the implementation platform [48]. The described performance simulation is primarily targeted at maximising implementation efficiency, using simple traffic models in the SDL environment.

The implementation platform described [49, 50] consists of an off-the-shelf IEEE 802.11 compatible MAC-less radio card, interfaced with a custom MAC module to a Windows NT Personal Computer (PC). The MAC module contains a Digital Signal Processor (DSP) which implements MAC functionality, along with an Field Programmable Gate Array (FPGA) which provides the host-MAC and MAC-radio interfaces. Appropriate hardware/software partitioning of the MAC is achieved through implementation of time-critical MAC functions in VHSIC Hardware Description Language (VHDL), while more complex functionality remains within the DSP.

The TUTWLAN development platform provides an excellent basis for a rapid prototyping system for wireless MAC protocols. The design flow allows for an iterative refinement from an initial specification through to a protocol implementation. However, other research such as that described by Heidemann et al. [51], illustrates the importance of achieving an appropriate level of detail in protocol evaluation. Too little detail can potentially give invalid simulation results, however the trade-off is that increasing detail adds complexity and

therefore required processing power. Transport protocols in particular are especially susceptible to MAC characteristics in a wireless environment [16, 52], so it is vital that care is taken in the evaluation of designed protocols. It is also obvious that mobility of nodes has impact on dynamic routing protocols at the network layer [53, 54].

It is not sufficient solely to provide suitable higher-layer models. The wireless physical layer has unique characteristics which can significantly affect MAC operation. Hence it is important that this aspect is given due respect in simulation. Ideally a development environment will allow realistic simulation early in the design lifetime. This allows the significance of design decisions to be objectively assessed before too much time is invested in what may be a poor solution to the given problem.

Although the hardware architecture of TUTWLAN is suitable for basic evaluation of wireless protocols, its form-factor restricts its use in real implementation scenarios. An implementation platform suited to use in mobile nodes and nodes with potentially restricted accessibility would facilitate a more thorough evaluation of a developing protocol.

Kotz et al. [17] suggest several ways in which the Mobile Ad-hoc Network research community can improve the quality and therefore productivity of their wireless protocol research. Among other things they encourage: use of consistent code between simulation and implementation allowing direct comparison of the results; use of appropriate models for the wireless physical layer in the target application; and flexible development environments that allow exploration of the effect of model parameters on design performance. The ideal wireless protocol development platform will provide a design and evaluation environment which meets these needs from specification through to implementation.

2.1 Modern Data-Link Protocols

Tanenbaum [55] defines the roles of the data-link layer as to:

- Provide a well-defined service interface to the network layer.
- Deal with transmission errors.
- Regulate the flow of data so that slow receivers are not swamped by fast senders.

This definition matches well with reality in a wired context, however the case of the wireless data-link layer is much more complicated. Regardless of physical layer, the principle service provided by the data-link layer is, of course, the transfer of data from the network layer on the source to the network layer on the destination machine. Error control is particularly applicable in a wireless network, as is data flow control. However two other roles can be added to the above list in the wireless case:

- Control of the physical layer to ensure optimum service as visible at the interface to the network layer.
- Support enhanced features as required by higher layers such as...
  - QoS: Bandwidth reservation, Traffic prioritisation
  - Security: Authentication, Privacy
  - Mobility: Handover between Access Points (APs)

To further clarify the roles of the data-link layer it is useful to consider in more detail the roles of the adjacent layers in the OSI protocol stack: the physical and network layers.

The Physical Layer

The key role of the Physical Layer (PHY) is to abstract signalling specific detail from a technology, and provide a bit delivery service to higher layers. This layer may also provide some framing to provide a basic packet broadcast service, as in the case of the IEEE 802.11 Physical Layer Convergence Protocol (PLCP). It is within the scope of the physical layer to provide some means of tolerance to channel errors – for example redundancy in symbol encoding. However,

there is generally no fixed requirement for the maintenance of a certain bit or packet error rate.

The interface between the physical and data-link layers may be quite complex. The MAC sub-layer is often designed to take advantage of – and therefore required to control – specific physical layer behaviours such as frequency hopping, spreading codes used in CDMA protocols, or any number of other parameters. This control may be required on a per-packet or even sub-packet scale, meaning that synchronisation of timing between MAC and PHY is essential.

The Network Layer

The network layer provides end-to-end delivery of packets. It relies on the data-link layer providing a semi-reliable host-to-host packet transfer service, and expands on this by providing for routing and fragmentation of packets. The most common example of a network layer protocol is the ubiquitous Internet Protocol (IP). There is no requirement that the network layer provide a reliable service. For example, IP provides what is termed as a 'best effort' packet delivery service, with error detection information applied only to the packet header. Duplicate or out-of-order packets are possible, as is data corruption.

The typical interface between the network and data link layers is similar to the physical layer interface previously discussed. The primary function of this interface is to facilitate the transfer of IP datagrams between peer TCP/IP stacks. Features such as QoS also require interaction between these layers, however current implementations of this typically involve operating system supplicants which manage this functionality.

In order to understand the facilities required of a wireless protocol development framework it is beneficial to consider existing relevant protocols. Three modern wireless protocols are considered below, with applications in Local Area Network (LAN), Personal Area Network (PAN), Metropolitan Area Network (MAN), and cellular network topologies. These are selected to be representative of the types of protocol that are of interest to most current researchers in this field. Each protocol is introduced, and set in the context of its typical application domain. The core medium access control method is discussed, along

with techniques used for error and flow control at the MAC layer. Advanced features supported by the MAC such as QoS and security are discussed. The procedures which govern the operation of the protocol are also covered. Finally, focus is given to aspects which have onerous timing requirements that may require hardware support.

2.1.1 IEEE 802.11

The IEEE 802.11 standard [2] defines the most common WLAN currently in use[1]. This technology applies packet broadcast radio to the licence-exempt Industrial, Scientific and Medical (ISM) frequency bands, to provide relatively low cost, and increasingly high bandwidth network communication. The initial physical layers defined were an infrared specification, along with Frequency Hopping Spread Spectrum (FHSS) and Direct Sequence Spread Spectrum (DSSS) radios. Only the DSSS physical layer was widely implemented, providing raw bandwidth of up to 2Mbps. Later amendments to the protocol have provided for bandwidths of 11Mbps [56], and 54Mbps (also supporting the 5GHz ISM band) [57, 58, 59]. The forthcoming 802.11n, expected in March 2009, will use Multiple Input, Multiple Output (MIMO) techniques to provide rates up to 540Mbps.

The core 802.11 protocol uses the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) medium access control method. When a station wishes to transmit, it first senses the channel for a period to determine if any other transmission is in progress. This carrier sense may be performed using an in-band Received Signal Strength Indication (RSSI) or carrier acquisition. If the medium is determined to be idle, then the transmission may begin. If the medium is busy when sensed, then the station begins its back-off procedure. The 802.11 back-off procedure requires that the station select a random back-off time up to the size of its *contention window*, and then wait until the medium has been (cumulatively) idle for that period of time before attempt-

[1] IEEE 802.11 technology is often referred to as *Wi-Fi*, however this term is more correctly used in reference to 802.11-based equipment which has passed interoperability tests specified by the Wi-Fi Alliance, and is therefore Wi-Fi Certified.

ing to transmit again. The contention window is set to a minimum value after each successful (acknowledged) transmission, and is doubled in size (up to a maximum) after each unsuccessful attempt. As well as the physical carrier sense, 802.11 also incorporates a virtual carrier sense mechanism. Frames can contain a duration field which informs listening stations of the reservation of the medium for the period of the transaction. Request to Send (RTS)/Clear to Send (CTS) exchanges attempt to address the hidden node problem [60, 61], and provide for floor acquisition to reduce the potential impact of collision for larger frames.

All directed frames that are received successfully must be immediately positively acknowledged by the receiving station. If the initiating station does not receive an acknowledgement within a certain period from transmission, then it will consider the transmission unsuccessful and will retry up to a certain number of times. Transmissions of certain frames such as Acknowledgement (ACK) and CTS are not required to follow the normal CSMA rules. Instead contention is avoided by requiring these frames be transmitted after a fixed duration, known as the Small Inter-Frame Space (SIFS), from the last received symbol of the prior packet. Successful reception of a frame is verified using a 32-bit IEEE CRC which is calculated over, and appended to, each frame transmitted. To meet the SIFS turnaround, hardware support for CRC verification is generally required. The CRC computation process can generally share this hardware, significantly reducing the burden on the MAC processor.

A set of communicating 802.11 stations is referred to as a Basic Service Set (BSS). Two specific network configurations are possible: an Independent Basic Service Set (IBSS) denotes a set of stations which communicate directly in what is often called an *ad hoc network*; alternatively, an Extended Service Set (ESS) consists of one or more stations, and an Access Point (AP) which provides a bridge between the wireless network and the Distribution Service (DS) which is typically (but not necessarily) a wired network[1]. Within an ESS configuration, each station must perform authentication and association procedures with the access point before communication may begin. Several authentication methods

[1]The specification is ambiguous regarding the correct use of the terms BSS and ESS. The convention taken here reflects the typical use of these terms within the 802.11 community

are specified using multi-stage transactions to reach the authenticated state. Once authenticated, a station may associate to the access point using a simple handshake, and begin exchanging data. Low power modes are supported where stations arrange to be awake only at specific times to exchange data with other network participants. These periods are marked by frames called beacons which are sent by the access point in an ESS.

Several other amendments provide features which contribute to the effectiveness of systems built on 802.11 technology. QoS facilities are defined in 802.11e [62], with provision for prioritisation of certain classes of traffic, admission control to QoS data streams, optimised frame exchange sequences, and improved power management facilities such as Automatic Power-Save Delivery (APSD). Security features are defined by 802.11i [63], which makes use of the Advanced Encryption Standard (AES) cipher to provide message confidentiality, integrity, and source authentication. Both 802.11e and 802.11i involve significant changes within the MAC sub-layer, including addition of information elements within MAC management frames. In particular, 802.11e makes changes to the collision avoidance back-off algorithm to prioritise certain classes of traffic. The QoS extensions defined in 802.11e provide differentiated services through different size Inter-Frame Space (IFS) and contention windows for different classes of traffic. An optional feature allows block acknowledgement of multiple directed frames. This will typically require modifications to the hardware prescribed above. The security features defined by the original 802.11 standard and the 802.11i amendment provide for encryption of frame payloads using various encryption techniques. Although software implementations of these features are possible, performance is likely to be severely affected unless dedicated hardware support is provided.

2.1.2 Bluetooth

Bluetooth [4] is an example of a WPAN. Its key attributes are an extremely low power consumption, excellent interoperability between products (enforced by a robust product qualification process), and low cost per node. The predominant use of the technology is in cable-replacement, with major applica-

tions including connectivity between mobile phone and headset, and Personal Digital Assistant (PDA) synchronisation. Bluetooth utilises the 2.4GHz ISM frequency band through an FHSS physical layer using Gaussian Frequency-Shift Keying (GFSK) to provide a nominal serialisation rate of 1Mbps. The Enhanced Data Rate (EDR) extensions to the protocol add the use of Phase Shift Keying (PSK) modulations to provide 2 and 3Mbps rates.

Bluetooth uses a TDMA method to share the medium between devices. The simplest Bluetooth network configuration is called a piconet. A piconet consists of exactly one device which is designated as the master, and one or more devices which are designated slaves. In a Bluetooth piconet, time is divided into slots of length 625μs. Piconet timing is maintained by the master, and all transmissions within the piconet must be aligned to the start of a slot. Master transmissions must start in an even-numbered slot, while slave transmissions must start in an odd-numbered slot. Various packet types are defined, with maximum durations of one, three and five slots. Slave devices may only transmit when addressed by the master. An exception to this rule is when a logical synchronous connection has been established such that certain slots are reserved for the up-link data transfer. The Bluetooth Synchronous Connection-Oriented (SCO) logical transport provides for reservation of slots for a given connection. This is the approach used to provide a level of quality of service to allow voice connections with fixed bandwidth and low latency. On the physical layer, transmissions follow a pseudo-random frequency hop sequence that is specific to the piconet master's device address, and the slot clock. This method ensures that multiple piconets may coexist without significant degradation of performance.

For each packet header an 8-bit Header Error Check (HEC) is provided to protect its integrity. The error control scheme (if any) applied to the packet payload depends on the packet type. In the case of SCO traffic, no MAC-layer error control is provided over the packet payload reflecting the relative insignificance of minor data-loss caused by random interference – a possibly audible blip in the speech. In contrast, Extended Synchronous Connection-Oriented (eSCO) packets use a 16-bit CRC over the packet payload, and an Automatic Repeat Request (ARQ) scheme for error recovery.

The Bluetooth Link Manager (LM) is responsible for controlling all aspects of the connection between two devices. The Link Manager Protocol (LMP) messages are carried using the Asynchronous Connection-Oriented (ACL) logical transport. Procedures such as connection establishment, authentication, and negotiation of low power modes all fall into the domain of the link manager. Like 802.11 the Bluetooth low power modes are based around an arrangement to awake at a certain time to exchange data. Data privacy is also provided at the link manager level, allowing the encryption of packet payloads.

Bluetooth has a much lower performance than the other protocols discussed here and so can typically be less reliant on dedicated hardware assistance. It does, however, have strict requirements around the timing of transmissions at slot boundaries. The specification dictates a clock drift of no more than 20ppm relative to the 625μs slot. Instantaneous timing may vary no more than 1μs from average, and at the receiving station a receive window of 10μs either side of the expected slot boundary is employed to provide tolerance for clock drift. In the final slot of a Bluetooth packet transmission, only 366μs of the slot is allowed to be used, leaving 249μs (625μs slot - 366μs activity - 10μs receive window) to allow transition of the radio between transmit and receive modes, and channel change to support the Bluetooth frequency hopping.

2.1.3 IEEE 802.16

The IEEE 802.16 [5] protocol is optimised for providing 'last-mile' wireless broadband access to relatively immobile subscriber stations. The Worldwide Interoperability for Microwave Access (WiMAX) Forum [6] is the primary promoter of this technology, and defines interoperability certification profiles much like the Wi-Fi Alliance does for IEEE 802.11. While 802.16 (certified as WiMAX) is targeted at fixed broadband access, the 802.16e amendment [64] extends the application of the protocol to mobile terminals. Growing technologies such as Mobile WiMAX and WiBro are based on the IEEE 802.16e standard. A range of physical layers are defined by the 802.16 standards for operation in the 2-11GHz or 10-66GHz using either Frequency Division Duplex (FDD) or Time Division Duplex (TDD) to support data-rates of up to 70Mbps.

Current certification profiles for Mobile WiMAX [65] specify the use of TDD in a range of bands from 2.3GHz to 3.8GHz.

The core MAC protocol uses a TDMA method controlled by the Base Station (BS) to share the medium between stations. Every fixed-length frame duration (5 milliseconds in Mobile WiMAX), the base station broadcasts a down-link sub-frame which includes the schedule of reservations for the remainder of that frame, and down-link data directed to the subscriber (or mobile) stations. The up-link phase follows, with stations able to transmit within their allocated reservations. Reservations are controlled by the base station, and can be dynamically adapted to suit the requirements of traffic. A CRC is appended to each MAC Protocol Data Unit (MPDU) to allow integrity verification, and an ARQ mechanism provides retransmission. As with Bluetooth, hardware support is required to enable the timing demands of the 802.16 TDD MAC protocol to be met. Other timings enforced by the specification are less onerous than the previously discussed protocols, though hardware assistance can significantly improve the performance of an implementation.

The 802.16e amendment defines mechanisms to support operations critical for mobile operations such as hand-off of stations between base stations, and low power modes. These mechanisms are implemented by state machines which communicate using messages passed through the MPDU delivery service. A security sub-layer provides authentication and encryption services for 802.16 stations. Authentication allows control of station access to network resources, and encryption provides a degree of privacy for packet payloads.

All data exchanged over an 802.16 link is associated with a connection, which defines a unidirectional flow of data between an 802.16 Base Station (BS) and a Subscriber Station (SS) (or Mobile Station (MS) in the case of 802.16e or Mobile WiMAX). Each connection may have specific service parameters, and the BS may alter scheduling to meet QoS requirements. Stations may request more bandwidth for a connection using either an existing reservation, or one which is specifically used for contention-based requests. Security features of 802.16 are similar in structure to those of 802.11 - as with that protocol, dedicated hardware is not necessary, but highly desirable for acceptable performance.

2.2 Implications for the Design Environment

Though the protocols discussed above seem quite distinct at first glance, it can be seen that they draw on a similar set of mechanisms to provide services tailored to their respective application areas. These common factors will provide some guidance in the design of a development framework.

All three protocols provide what can be considered as a packet delivery service between peer MAC sub-layers. Each makes use of some form of error control code such as a CRC, which is computed across some portion of the transmitted packet. Similarly, the encryption facilities require the processing of packet data. ARQ protocols are employed in all three examples primarily to increase link reliability.

Each protocol has aspects which require precise timing of activity: IEEE 802.11 must schedule acknowledgement packets for a SIFS duration after reception of a directed packet; Bluetooth requires synchronisation of transmissions to the slot boundary; and IEEE 802.16 requires transmission at the instant specified by the base station. These features require a development framework which provides the ability to build protocol mechanisms with highly deterministic timing.

Slow (relative to physical-layer symbol timing) internal state machines provide the network control associated with each protocol, including procedures for the forming and breaking of associations or connections, and management of low-power states. Internal timers provide an important supporting role in these mechanisms and must be provided for in a development platform.

Chapter 3

Representation

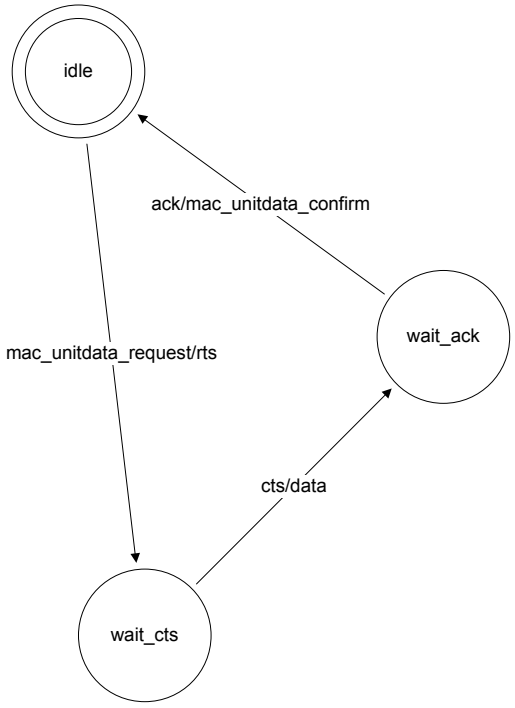
It is necessary to choose a design representation for use in initial specification of protocols within the development framework. The suitable representation will not only allow capture of the range of protocol functionalities identified in Chapter 2, but will also support the idealised design flow discussed in Chapter 1. By providing for a methodical design flow the quality of the design output is increased [66]. A key factor in this is allowing a top-down design approach [67] involving successive refinement of the representation. Design modularity will improve tracking of errors and allow complexity management for large specifications. The chosen representation will preferably enable rapid exploration of the various design alternatives which may potentially fit the requirements of a given initial specification.

The ability to efficiently and effectively describe the functionality that typically comprises the MAC sub-layer is an essential attribute in the candidate representations. As identified in Chapter 2, the range of functionalities present in such subsystems is quite diverse. It is possible, however to generalise typical MAC operations into the following categories:

- Slow (relative to system clock) internal communicating state machines.
- Timer manipulation.

- Communication with the host and the radio.
- Data transfer (generally trying to minimise copy).
- Short bursts of intense computation – e.g. for error control, or encryption.
- Activities with strict real-time requirements.

Representation of the different aspects of functionality is best achieved in a variety of ways. Graphical design techniques are extremely effective for the description of state machines and relatively simple algorithms. Design representations such as flow-charts and state transition diagrams allow design and visual analysis of these elements in a 'user-friendly' form.




```
void state_machine(int input)
{
  static int state = STATE_IDLE;

  switch(state)
  {
    case STATE_IDLE:
      if (input == SIGNAL_MAC_UNITDATA_REQUEST)
      {
        output(SIGNAL_RTS);
        state = STATE_WAIT_CTS;
        break;
      }
      break;
    case STATE_WAIT_CTS:
      if (input == SIGNAL_CTS)
      {
        output(SIGNAL_DATA);
        state = STATE_WAIT_ACK;
        break;
      }
      break;
    case STATE_WAIT_ACK:
      if (input == SIGNAL_ACK)
      {
        output(SIGNAL_MAC_UNITDATA_CONFIRM);
        state = STATE_IDLE;
        break;
      }
      break;
  }
}
```

Figure 3.1: Graphical and Textual State Machine Representations

Figure 3.1 shows the Mealy machine state transition diagram [68] for the transmit sequence of a simple RTS/CTS-based protocol, along with an equivalent description coded in C. It can be seen from this example that the graphical representation not only conveys all information necessary for understanding of

the machine, but also presents it in a form which allows the viewer to quickly grasp the essence of the problem rather than grappling with the syntax and verbosity of a textual language.

Graphical techniques maintain some relevance when applied to algorithmic operations, though as complexity increases the representation can become quite cumbersome. Simple algorithmic operations are often best represented in a textual form, as illustrated by the example checksum computation in Figure 3.2.

```
uint16 checksum(char *frame, int len)
{
  uint16 sum = 0;

  while (--len)
    sum = sum + frame[len];

  return sum;
}
```

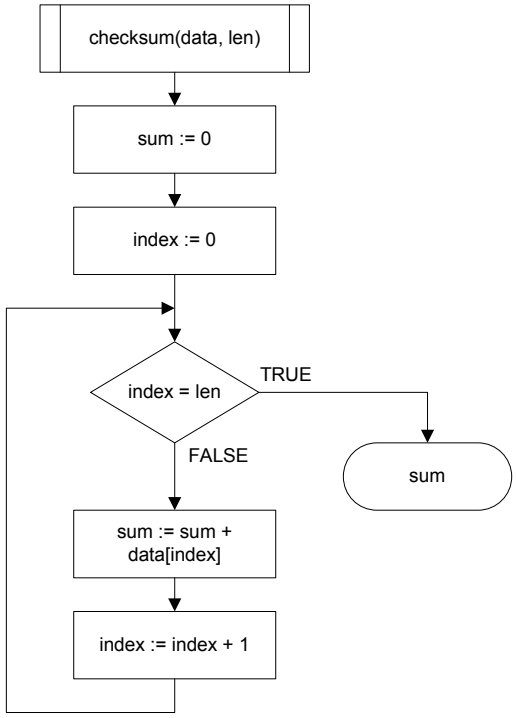


Figure 3.2: Textual and Graphical Implementations of a Checksum

These different methods of representation also have varying levels of relevance at different stages of the design process. Initial design of a protocol typically involves specification of the state machines which define its basic operation. As the process progresses, focus shifts to adding detail such as Protocol Data Unit (PDU) format, timing, and specific algorithms for aspects such as error control, security services, or even optimisation of naively implemented functionality.

A MAC protocol is a real-time distributed algorithm, whose fundamental role is to control access to the communication channel. Much research has been

directed into the more general area of distributed algorithm design. It is well known in this field of research that the use of formal methods in the design of such systems results in significantly higher quality software [13, 14, 15, 69]. Techniques such as Unified Modelling Language (UML), state-charts and Z, have been proposed as means of migrating from user requirements specification towards an implementation. Such techniques often have a formal mathematical foundation, which allows verification of the correctness and completeness of a specification.

These methods have significant relevance to the challenges presented by MAC protocol design. The often-strict timing requirements of MAC protocols mean they are commonly implemented at a low level – in hardware, firmware, or embedded software and have a significantly longer design cycle than typical software applications [70]. For this reason, any process which may reduce the number of iterations of the cycle is justified. In addition, MAC protocols can have reliability requirements equalling or exceeding those of life-support application software. Use of formal description techniques has the potential to reduce the number of design-cycle iterations required, as well as increase the reliability of the designed protocol.

Despite the benefits of formal methods in protocol design being well-known in academia, uptake in industry has been relatively slow [13]. Even in academia, a minority of research efforts in the field of wireless MAC protocols has made use of formal methods. This is due in a large part to the lack of proven tools and design flows that would make productivity increase apparent. For this reason it is desirable to support a given design flow from an abstract specification through to an implementation with a large degree of automation. The ultimate goal for the design flow is to provide the protocol designer with a progressive modelling process, from the abstract towards implementation. We discuss several Formal Description Techniques (FDTs) in Section 3.2, but first introduce a protocol which is used as a common example.

3.1 The PTPMAC Protocol

To demonstrate the effectiveness of the framework described within this thesis, we use a case study of an example protocol design process. This protocol is rather unimaginatively named the Point-To-Point Medium Access Control protocol – PTPMAC. The PTPMAC protocol is briefly introduced here, and developed through the remainder of the text.

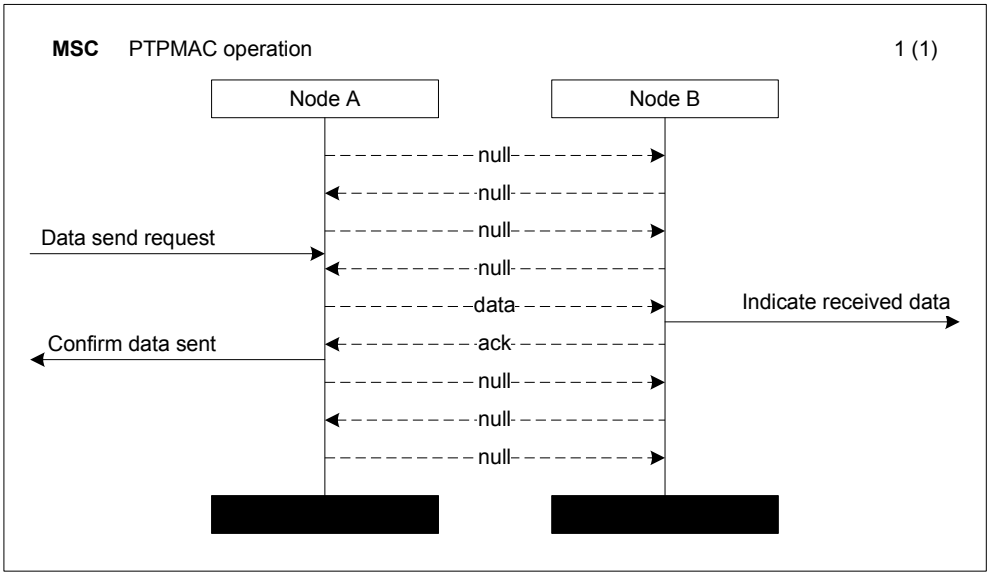
The intended application of the PTPMAC protocol is to provide a reliable data transfer service across a broadcast medium shared by only two stations – a point-to-point link. Such links are often used to provide backbone connectivity. The design objectives are as follows:

Basic Function: The primary objective is to provide a point-to-point data transfer service for IEEE 802.3 or similar packets across a wireless medium.

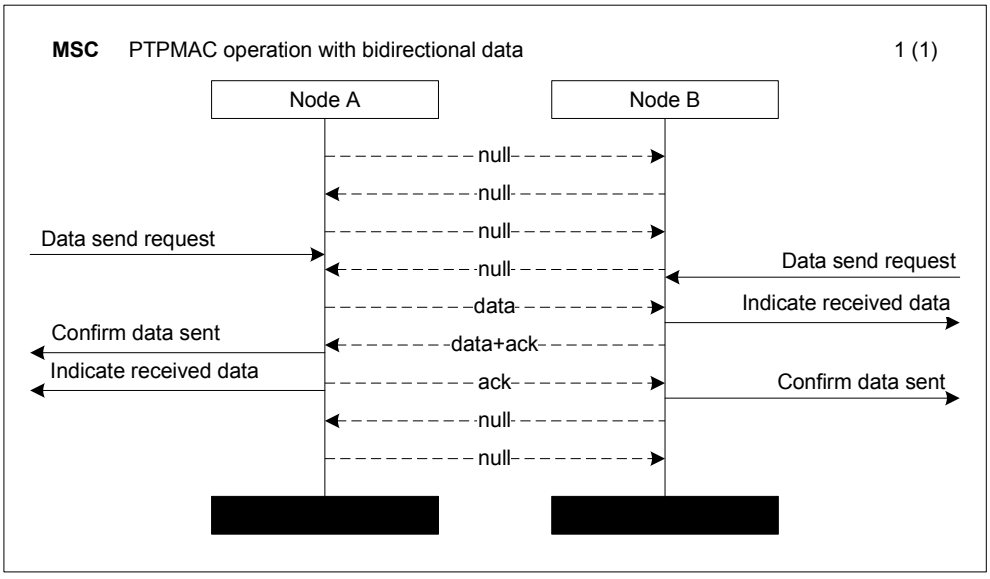
Throughput: The protocol shall attempt to maximise throughput given a 802.11b physical layer. It is believed that the target physical layer in the implementation will cause the throughput to be channel-bandwidth-bound rather than processor-bound. Hence this objective requires that frame encoding and exchange sequences are made as efficient as possible.

Reliability: The wireless medium is inherently unreliable. The protocol must be tolerant of data loss and corruption at the physical layer. It is desirable that corrupted data is detected within the MAC layer, and not passed to higher layers of the network stack.

The chosen approach for PTPMAC is based around the idea of token passing. The two stations which comprise the point-to-point network take turns transmitting a token to each other. The token may include acknowledgement of the previous correctly received data, some amount of data, both or neither. The basic frame exchange sequence of the PTPMAC protocol is shown in Message Sequence Chart (MSC) form in Figures 3.3(a) and 3.3(b), with uni and bi-directional data respectively. The MSC provides an effective graphical representation of communication between entities within the system.



(a) Unidirectional Data



(b) Bidirectional Data

Figure 3.3: PTPMAC Basic Operation

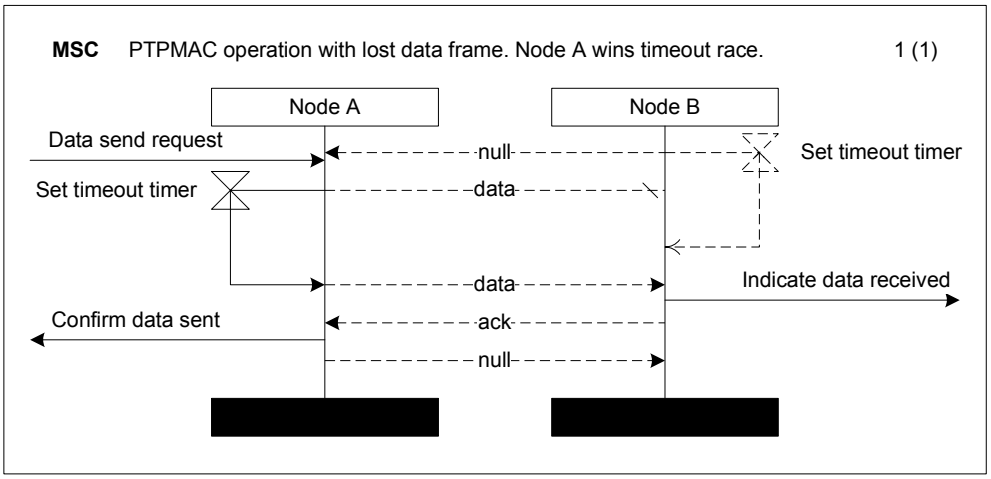


Figure 3.4: PTPMAC Recovery from Data Loss

Each vertical line represents an entity – in the case of Figures 3.3(a) and 3.3(b) each is an instance of the MAC. Horizontal arrows show the sequence of message exchanges with time progressing from the top to the bottom of the diagram.

In the PTPMAC protocol a timer which is set on transmission provides a facility for recovery from a lost token. If this timer expires prior to reception of the token, then the previous transmission is repeated. This is illustrated in the MSC in Figure 3.4. If a received token does not indicate acknowledgement of data which was associated with the previously sent token, then that data is retransmitted within the next sent token. Formalisation of the proposed mechanism into an MSC exposes a race condition between the response timeout timers of the two communicating nodes. This requires that some random variation of the timeout is applied to avoid retry collisions.

3.2 Formal Description Techniques

The choice of a FDT is an important decision in formulating the design flow. Factors such as availability of tools, suitability for the problem domain, and likely learning curve must be taken into account [71, 72]. The ability to design

graphically is a significant advantage in describing distributed communications systems. Message sequence charts, and state-transition diagrams are good examples of visual representations that are generally intuitive to network protocol designers, and techniques that are easily related to these are likely to be good candidates for use. Three formal description techniques are discussed here as candidates for use in this work: E-LOTOS, ESTELLE, and SDL.

The PTPMAC protocol introduced in Section 3.1 is used as a common reference example in the discussion of these techniques. In presenting the protocol, reference is made to six specific messages or *service primitives* which define the API for the MAC and PHY services. These are: `mac_unitdata_request`; `mac_unitdata_confirm`; `mac_unitdata_indication`; `phy_data_request`; `phy_data_confirm`; and `phy_data_indication`. Proper definition of these can be found later in Section 4.1, however for now it suffices to say that they are the primary messages by which the MAC will communicate with its environment.

We begin coverage of FDTs with an introduction to E-LOTOS.

3.2.1 E-LOTOS

LOTOS – the *Language Of Temporal Ordering Specification* – was defined by the International Organization for Standardization (ISO) as a means for producing formal descriptions of the OSI services and protocols. LOTOS was first standardised in 1988 as ISO-8807. In 1993, the ISO proposed that enhancements be made to LOTOS, in order to maintain its relevance as a formal specification language. In 1998 E-LOTOS – *Enhancements to* LOTOS [73] – was submitted as an ISO standard. E-LOTOS enhances LOTOS in a number of ways while maintaining a similar structure. The most significant of the enhancements are:

- allowing the concept of quantitative time. LOTOS can only model the temporal ordering of interactions.
- use of a more user-friendly data type language, and addition of pre-

defined types.

- providing for modularity, and parameterised modules to improve code re-usability.
- the addition of an exception model.

The concepts involved in E-LOTOS represent a significantly different approach to the description of a system to that taken in the two other techniques discussed in this chapter (ESTELLE and SDL). Rather than implicitly describing the behaviour of a system by constructing another system that behaves in the same way, E-LOTOS instead allows the specifier to explicitly describe the behaviour in terms of correct sequences of interactions with the environment. E-LOTOS does not provide a graphical representation, however derivation of a MSC is possible from the textual notation [74].

Due to the lack of a graphical representation, and having a design paradigm significantly different from the commonly employed imperative programming languages, E-LOTOS presents a steep learning curve for the typical protocol engineer. Its form however, does provide advantages in formal verification of designs. E-LOTOS is considered here for this reason, along with the benefit of discussing a contrasting approach to formal description using ESTELLE and SDL.

Data-types, Variables, and Functions

Though our primary concern is the representation of behaviour, a key component of this is manipulation of variables within a specification. E-LOTOS provides functionality for defining new data-types for variables within a specification. We briefly introduce these facilities here, before discussion of behaviour modelling in E-LOTOS.

Type declarations within E-LOTOS are either *type synonym* or *data-type* declarations. The simplest form of type synonym is the renaming of a type as shown in the example below:

```
type seqnum_t is integer endtype
```

In a similar way, record types can be defined. The example below shows the definition of an MAC Protocol Data Unit (MPDU) type for use in the E-LOTOS representation of the PTPMAC protocol.

```
type mpdu_t is
  (ack => ack_t,
   ackseq => seqnum_t,
   data => boolean_t,
   dataseq => seqnum_t,
   dataref => dataref_t)
endtype
```

In contrast, *data-types* define entirely new types. Typically this involves a union (denoted using the | separator) of the various type constructors. Consider the definition of a simple (effectively boolean) type for representation of an acknowledgement bit:

```
type ack_t is ACK | NACK endtype
```

Behaviour and Communication

The E-LOTOS FDT allows systems to be represented in terms of their allowable interactions with the environment. Allowable sequences of interactions are described by way of what is known as a *behaviour expression*. These expressions can be represented graphically as trees, with each node indicating a state of the system, and each vertex corresponding to an interaction. Figure 3.5 shows an example E-LOTOS behaviour tree.

Interactions in E-LOTOS are synchronous, instantaneous and atomic interactions. They occur between *processes*, and take place through *gates*. They are

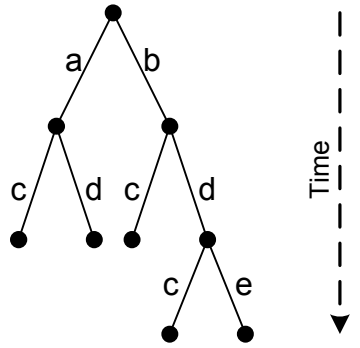


Figure 3.5: E-LOTOS Behaviour Tree

sequential – two distinct interactions cannot happen simultaneously. As these events are synchronous – that is, they can only occur with the cooperation of all processes involved – an interaction is sometimes described as two processes *synchronising on a gate*. An interaction may involve the offer or acceptance of data, and processes may make a decision on whether or not to take part in an interaction based on the value or type of data that is offered. The behaviour expression which describes an interaction through the gate `phy_data_request` which offers the variables `txparams` and `txmpdu` would be written in E-LOTOS as:

```
phy_data_request(!txparams, !txmpdu)
```

Correspondingly, the expression which would match an interaction through gate `phy_data_indication` with two parameters of the same types as variable `rxparams` and `rxmpdu` is shown below. This expression causes the parameters of the interaction to be stored into these variables.

```
phy_data_indication(?rxparams, ?rxmpdu)
```

The *action prefix* operator – ‘`;`’ – provides a means of ordering events, by asserting that a certain event must occur before the following behaviour expression. Using this operator we can begin to describe the permissible sequences of events. As an example, consider the combination of the `phy_data_request` interaction mentioned above (causing a physical layer transmission to start), and the `phy_data_confirm` interaction (which indicates the completion of transmission):

```
phy_data_request(!txparams, !txmpdu); phy_data_confirm
```

For interactions which do not specify the offer or acceptance of data (as in the `phy_data_confirm` example above), it is not clear whether the message has been 'sent' or 'received'. This, however, is irrelevant due to the fact that interactions are synchronous in E-LOTOS. Both the 'sending' and 'receiving' processes are required to synchronise.

In a real system, many behaviours are possible depending on the stimulus (interactions) provided by the environment. The *choice* operator – '`[]`' – allows for alternate behaviours to be expressed. Visually this is indicated by multiple children of a node in a behaviour tree. Consider the expression below where any offer or acceptance of data has been removed for brevity.

```
(mac_unitdata_request; phy_data_request; phy_data_confirm;
 phy_data_indication; mac_unitdata_confirm) []
(phy_data_indication; mac_unitdata_indication)
```

This expression describes either the transmission or reception of a data unit. The behaviour will be dictated by whether a `mac_unitdata_request` (presumably from the host) or an `phy_data_indication` (presumably from the physical layer) is the first interaction to occur.

As previously mentioned, E-LOTOS models systems in terms of their *observable behaviour*, as seen by the environment. However a system whose behaviour is defined purely in terms of its sequences of interactions with the environment, would be constrained to being deterministic. In real-life systems, non-deterministic behaviour (as seen by the environment), is a common occurrence – often due to factors such as timeouts and finite system resource limits. The internal event (denoted `i`), allows modelling of such occurrences in E-LOTOS.

```
mac_unitdata_request; phy_data_request; phy_data_confirm;
((phy_data_indication; mac_unitdata_confirm) []
 (i; phy_data_request; phy_data_confirm))
```

The above behaviour illustrates a possible use of the internal event. In this case, `i` represents a transmission timeout. If the `phy_data_indication` event occurs before the timeout (`i`), then the transmission will complete as described in the previous behaviour expression. However, if the internal event occurs first, then the medium transmit sequence will be attempted again. This models a simple retransmission mechanism.

The *disabling* operator – '`[>`', as used in the expression $\texttt{B}_1$ `[>` $\texttt{B}_2$ – allows behaviour $\texttt{B}_2$ to disable behaviour $\texttt{B}_1$ if it has not successfully completed. In our application domain, this provides a way of explicitly modelling transmission timeouts.

```
mac_unitdata_request; phy_data_request; phy_data_confirm;
((phy_data_indication; mac_unitdata_confirm) [>
 (TimeoutExpires; phy_data_request; phy_data_confirm))
```

E-LOTOS provides a `time` data-type which acts as the basis of the support for modelling of real-time behaviour. The `wait` operator allows specification of a delay within a behaviour expression. This operator takes a single parameter (which may be variable) and delays for that duration. The simple example below adds detail to the above behaviour expression given presence of a variable named `txtimeout` which is of type `time`:

```
mac_unitdata_request; phy_data_request; phy_data_confirm;
((phy_data_indication; mac_unitdata_confirm) [>
 (wait(txtimeout); phy_data_request; phy_data_confirm))
```

Concurrency between behaviours can take various forms. Asynchronous concurrency of two behaviour expressions is modelled using the operator '`|||`'. The composite behaviour will synchronise only on termination. At the opposite end of the concurrency spectrum is the operator '`||`', which requires that the two behaviours synchronise on all communication. The example below shows the transmit and receive behaviour of the PTPMAC protocol split into

two expressions. The || operator requires that these two synchronise on all gates that they have in common, which in this case is solely received. Other degrees of synchronisation can be achieved in composite behaviours, including synchronising only on a given subset of the common gates, and synchronisation between more than two behaviours.

```
(mac_unitdata_request; phy_data_request; phy_data_confirm;
 received; mac_unitdata_confirm) ||
(phy_data_indication; received; mac_unitdata_indication)
```

Behaviour expressions in E-LOTOS are encapsulated within a *process* to form an entity with a given behaviour. The process declaration may specify any number of gates, input and output parameters, and a defining behaviour expression. An instantiation of a process is a valid behaviour expression in itself, so recursive instantiation of processes are often used to express repetitive behaviour. Consider the process named transmit_unit which is shown below. This process can communicate on five gates: mac_unitdata_request, mac_unitdata_confirm, phy_data_request, phy_unitdata_confirm, and received. The process also takes an input parameter named txtimeout which is of type time.

```
process transmit_unit[mac_unitdata_request,
                      mac_unitdata_confirm,
                      phy_data_request,
                      phy_data_confirm,
                      received]
                     (in txtimeout:time) is
  ((mac_unitdata_request) []
   (received; mac_unitdata_confirm) []
   (wait(txtimeout); phy_data_request; phy_data_confirm));

  transmit_unit[...](txtimeout)
endproc
```

The behaviour defined within this process builds on the previous examples to create a model of the PTPMAC transmit functionality. Recursive instantiation of the process means that the behaviour is continuous, unlike the simple expressions described previously.

Summary

Using the basic constructs discussed here, E-LOTOS allows the detailed specification of complex systems. Features of the language allow for modularisation and code reuse to improve usability in real-world applications. As discussed earlier in the section, however, it presents a steep learning curve for new users which is compounded by the lack of a standardised graphical representation.

Several tools are available for supporting LOTOS design. The most significant of these is the CADP toolkit [75] which includes facilities for translation of LOTOS specification behaviour to a C representation for analysis and verification. In contrast, very few tools support the E-LOTOS language. The TRAIAN compiler [76] provides for translation of an E-LOTOS subset known as LOTOS-NT into the C language, and the Simulator for E-LOTOS Specification (SEEL) [77] gives a simple graphical simulation environment for E-LOTOS specifications.

3.2.2 ESTELLE

The *Extended Finite State Machine Language* – ESTELLE was first standardised by the International Organization for Standardization (ISO) as ISO-9074 in 1989. It provides a means for formally specifying distributed, concurrent information processing systems, and hence is generally suited to protocol description (In fact, ESTELLE was initially created as a means for describing OSI services and protocols). As the name suggests, ESTELLE is based on the theory of finite state automata, extended using the Pascal language. Though the standard defines a textual language, Templemore-Finlayson [78, 79] has proposed a graphical representation known as ESTELLE/GR (in contrast to the textual or *phrasal representation* referred to as ESTELLE/PR).

ESTELLE/GR improves the usefulness of ESTELLE, by shortening the learning curve in some cases. Several language constructs, however, are un-catered for in ESTELLE/GR. We discuss the ESTELLE concepts here using a mixture of textual and graphical constructs as available in the primary ESTELLE/GR design entry tool which is described in [78, 79].

Data-types, Variables, Functions, and Procedures

The basic unit of functionality in ESTELLE – the *module* – extends the pure finite state automaton concept by providing for additional state in the form of *context variables*. If system state were required to be modelled solely by the automaton state (referred to as *major state* or *control state*), then the number of states would quickly become large. Context variables allow the automaton model to remain manageable when modelling real-world systems. Data-types and variables in ESTELLE follow from the Pascal origins of the language. New data-types are defined using the `type` keyword, and can be created as synonyms of existing types, records, or enumerated types. The `const` and `var` keywords allow definition of constants and variables respectively. Consider the example code below:

```
type seqnum_t = integer;
     ack_t = ( NACK, ACK );
     mpdu_t = record
                ack     : ack_t;
                ackseq  : seqnum_t;
                data    : boolean;
                dataseq : seqnum_t;
                dataref : dataref_t;
              end;

const min_timeout = 10000;
const max_timeout = 20000;

var rxmpdu : mpdu_t;
```

Functions and procedures can be used within ESTELLE as per standard Pascal. Functions accept parameters and return a result of some type. Procedures do not return a result, but contents of variables passed as parameters may be modified to achieve the same result.

Behaviour and Communication

The structure of an ESTELLE specification is represented by a hierarchical structure of entities known as *modules*, which are connected by *channels*. The ESTELLE module is based around a finite state automaton, which accepts inputs, and produces output on state transitions. Inputs and outputs to and from a module are known as *interactions*, and take place through *interaction points*. State *transitions* may be activated or *fired* by an input (in which case they consume that input), or may be *spontaneous* – occurring after a delay has passed, or when a condition becomes true. Modules may also be non-deterministic, with current state and input dictating a set of allowable transitions, only one of which will be taken.

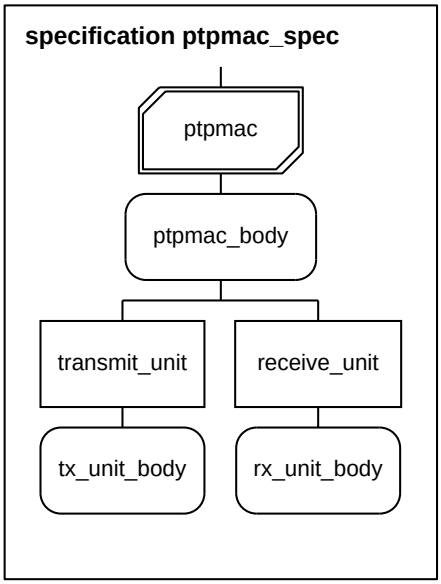


Figure 3.6: PTPMAC Module Hierarchy in ESTELLE/GR

Figure 3.6 shows the module hierarchy of an ESTELLE/GR specification of the PTPMAC protocol. The tree-like structure shows the encapsulation of the `transmit_unit` and `receive_unit` module headers (and their corresponding bodies) within the `ptpmac_body` definition. The interaction points of these modules are not visible within the ESTELLE/GR representation, but are defined in associated ESTELLE/PR. It is also necessary to textually define the external channels and the interactions which they will carry:

```
channel host_to_mac (host,mac);
  by host:
    mac_unitdata_request (dest:macaddr_t; data:dataref_t;
                          pri:priority_t; flags:flags_t);
  by mac:
    mac_unitdata_confirm (data:dataref_t; txstatus:txstatus_t);
    mac_unitdata_indication (source:macaddr_t; dest:macaddr_t;
                             data:dataref_t; pri:priority_t;
                             flags:flags_t);

channel mac_to_phy (mac,phy);
  by mac:
    phy_data_request (txparams:txparams_t; mpdu:mpdu_t);
  by phy:
    phy_data_confirm ;
    phy_data_indication (rxparams:rxparams_t; mpdu:mpdu_t);
```

The two channels defined here provide for exchange of MAC and physical layer service access point primitives (detailed later in Chapter 4) between the host and MAC, and the MAC and physical layer. Each channel defines two *roles* or endpoints – for the `host_to_mac` channel these are `host` and `mac`. A module which is connected to the channel must nominate one of these roles, and this dictates the interactions which this module may send over the channel. The definition of the `ptpmac` module header with its interaction points is expressed as:

```
module ptpmac systemprocess ;
  ip
    macsap : host_to_mac (mac) common queue ;
    physap : mac_to_phy (mac) common queue ;
end;
```

Each module header must have one or more module body definitions before it is useful. The module body definition contains three sections: declarations, initialisation, and transitions. Allowable declarations within a module body include a state list, as well as further definitions of modules and channels, and internal interaction points. The initialisation section allows initial values to be defined for variables, and an initial state to be specified. Finally, the set of allowable transitions are defined.

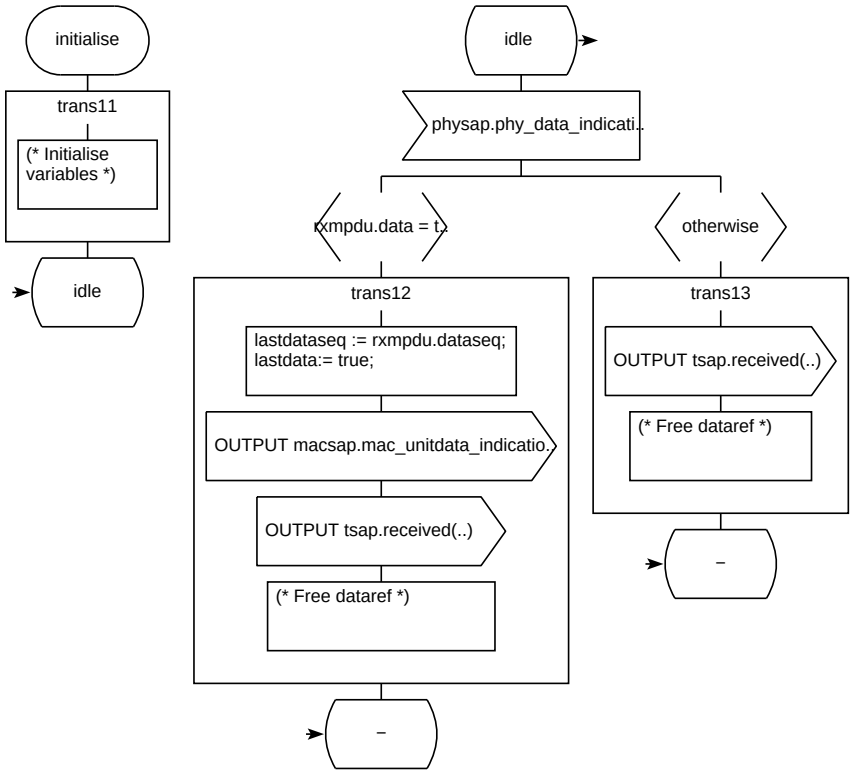


Figure 3.7: PTPMAC Receive Unit in Estelle/GR

Figure 3.7 illustrates the Estelle/GR description of the behaviour of the `receive_unit` module from the PTPMAC specification, and Figure 3.8 shows

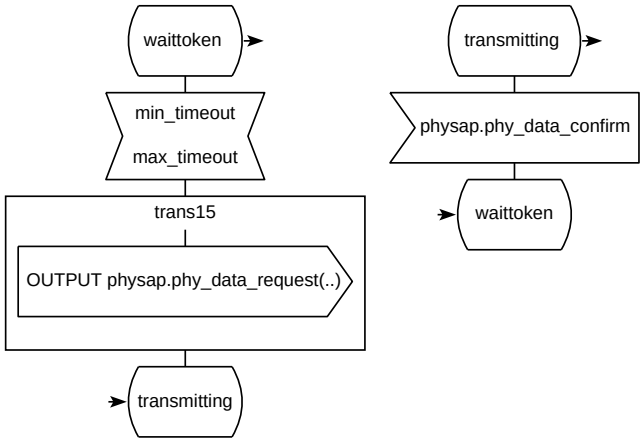


Figure 3.8: Partial PTPMAC Transmit Unit in ESTELLE/GR

part of the corresponding `transmit_unit`. Transitions in an ESTELLE automaton are enabled when a set of conditions are satisfied. These conditions may relate to: the state (control state or context variables); an interaction at the head of a queue; the state of timers (discussed below) or the transition priority . If no transitions are enabled, then the system remains static until this situation changes (due to either an externally initiated interaction, or a local timer expiry). Although multiple transitions may be enabled at a given time, only one of these can be taken. The firing of a transition is atomic. This means that intermediate values assumed by variables during a transaction are not visible outside the module. Transitions have no time associated with their execution, but ESTELLE allows a delay to be specified, which must elapse before a spontaneous transition may be enabled. This allows the modelling of real-time behaviour.

As mentioned, channels provide a path connecting two interaction points, over which a finite set of interactions can take place. ESTELLE provides two methods for establishing and breaking channels between modules, *connect/disconnect* and *attach/detach*. The process of connection refers to the creation of a channel which joins two external interaction points of children of the module. Disconnection is the reverse of this process. An external interaction point of a module may be *attached* to an external interaction point of one of its children. The reverse is done via *detach*.

The two varieties of channel establishment are said to *bind* interaction points. Correspondingly the reverse operations are said to *unbind* them. It is permissible to output interactions through an interaction point which is not bound – these interactions are lost. Interactions are received into queues, which are unbounded. A module can assign queues to each of its interaction points, or combine input interactions from multiple interaction points into a common queue. A module is oblivious to all but the interaction at the head of each queue.

All communication between ESTELLE modules is performed using either interactions, or shared variables. Interactions may be sent by a module at any time, and provide non-blocking communication. Parameters may be associated with a given interaction to allow communication of state. Alternatively, variables can be shared between a parent and child. To achieve this the child must declare the variable as *exported*.

Summary

The ESTELLE language provides sufficient constructs to support the representation of necessary functionality for MAC protocols developed within the targeted framework. Its primary weakness, however, lies in the ESTELLE/GR representation, and supporting tools.

The primary tool supporting ESTELLE/GR design entry is that described with the ESTELLE/GR language in [78, 79]. This editor allows entry of ESTELLE/GR specifications, and export of the associated ESTELLE/PR, and is included in the ESTELLE Development Tool-set (EDT) [80] along with analysis and compilation tools. This tool has major disadvantages in its reliance on hyper-linking between diagrams, and hiding of textual information. These factors mean that ESTELLE/GR descriptions are not able to be easily interpreted outside of this tool – for example in hard copy.

3.2.3 SDL

The *Specification and Description Language* – SDL was initially standardised by the International Consultative Committee on Telegraphy and Telephony (CCITT) as CCITT Recommendation Z.100 in 1976. Since then, the CCITT has become known as the International Telecommunication Union (ITU), and SDL specifications have continued to be updated, with the latest published in 2002 [81].

SDL is intended for providing "unambiguous specification and description of the behaviour of telecommunication systems". It began its life as an informal graphical representation, but has evolved into a formal technique with both graphical (GR - Graphical Representation) and textual (PR - Phrasal Representation) styles. SDL/GR and SDL/PR are equivalent, and automated translation between the two is possible.

Data-types, Variables, and Procedures

Like ESTELLE, SDL descriptions are based on the concept of communicating extended finite state machines. The basic finite state machines are extended by way of variables. SDL provides facilities for defining new data-types either from scratch, or by modifying existing ones. Data-types may be simple re-namings of existing types using the `syntype` keyword, or structures or enumerated types using the more general `value type` construction. The code below shows examples of these three methods of defining new data-types.

```
/* Type for the sequence numbers */
syntype seqnum_t = Integer;
endsyntype;

/* ACK field of the MPDU */
value type ack_t;
literals
  NACK, ACK;
```

```
endvalue type;

/* This defines the PTPMAC MPDU */
value type mpdu_t;
struct
  ack     ack_t;
  ackseq  seqnum_t;
  data    boolean_t;
  dataseq seqnum_t;
  dataref dataref_t;
endvalue type;
```

Constants can be declared in SDL using the `synonym` keyword with a specified data-type and value. Variables are declared using the `dcl` keyword, and optionally may have a default value specified.

```
/* Timeout range */
synonym min_timeout Integer = 10000;
synonym max_timeout Integer = 20000;

/* This variable stores the transmit MPDU */
dcl txmpdu mpdu_t;
```

A key feature of the SDL/GR representation is the well-defined method for including blocks of SDL/PR code. The ability to mix SDL/GR and SDL/PR allows the designer flexibility in choosing the most convenient method of representing any particular component.

Structure, Behaviour, and Communication

The basic design unit within an SDL description is the *agent*. An agent may contain any combination of variables, procedures, a state machine, and a set of contained agents. Agents can be further classified into *block agents* and

process agents, with the difference lying in execution scheduling of the state machines of that agent and any encapsulated ones. A block agent may contain any combination of block or process agents, which are executed concurrently. In contrast, a process agent may contain only process agents, whose execution is interleaved. Block and process agents can be defined as types which may then be instantiated any number of times allowing code reuse.

The *signal* is the basic unit of communication between SDL agents. Signals may include a list of parameters, in which case the data-types must be defined for that signal. Signals may be sent between agent instances, or between the environment and an agent instance. Each process instance has an unlimited buffer (known as the *input port*) which holds all received signals not yet consumed by the state machine.

For a signal to be exchanged between two agents, there must exist a path between those agents. A path may be implicitly defined through the containment of one agent by another, or may be explicitly defined as a *channel*. Channels may be declared to be uni-directional or bi-directional, and may be constrained to carry only certain signals. They maintain the ordering of signals they carry. SDL also includes the concept of *remote variables* which effectively provide a short-hand notation for a signal exchange to convey a variable value.

Figure 3.9 shows the internal structure of the SDL block agent which defines the PTPMAC entity. Two internal process agents are depicted (`transmit_unit` and `receive_unit`), which are connected by channels to the `macsap` and `physap` gates of the `ptpmac_mac` block. It can be seen that the graphical description of the channels includes definition of the signals which they may carry. These signals are defined textually in the body of a text symbol within an SDL specification. Parameters may optionally be included in the signal definition as a comma-separated list of data-types surrounded by parenthesis after the signal name. Visible in Figure 3.9 is the definition of the internal `received` signal which carries a single parameter of type `mdpu_t`.

These elements of SDL facilitate the top-down design methodology. Specification structure and subsystem interfaces may be designed first, before internal

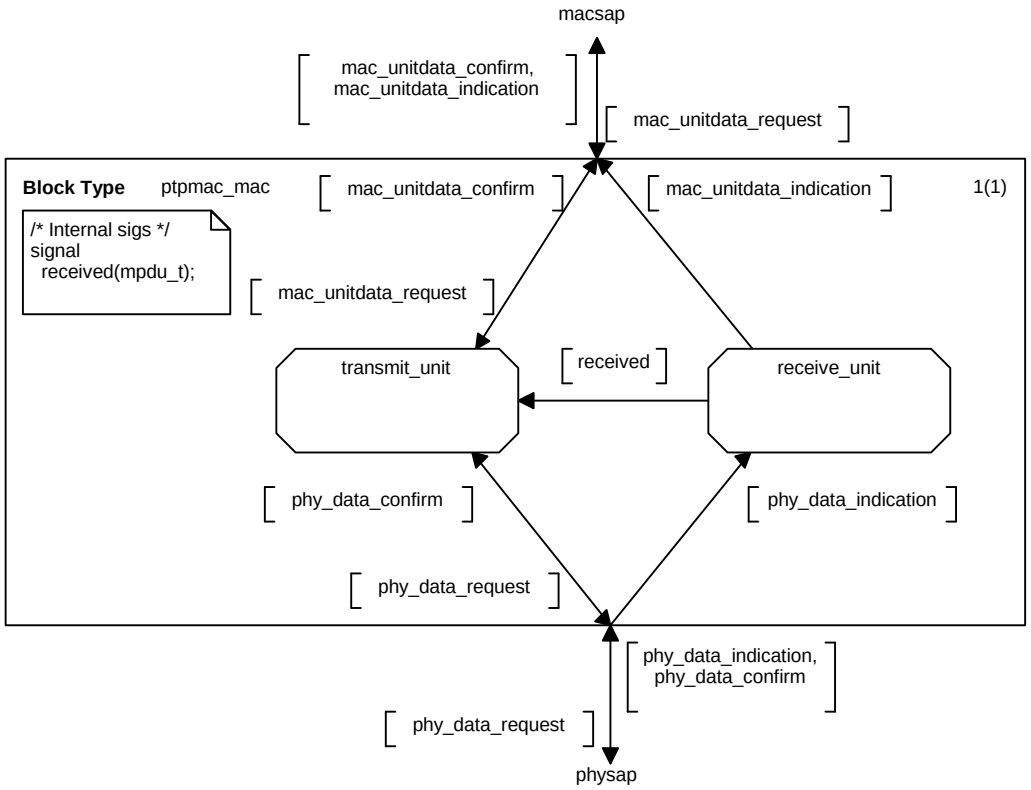


Figure 3.9: PTPMAC Internal Structure in SDL

agent functionality is implemented. Functionality is provided in SDL through finite state machines (FSMs), which may be included in either block or process agents. Within the state machine of an agent, behaviour is modelled through *states* and *transitions*. A *state* represents an agent context in which a signal may be consumed. The consumption of the signal by an *input* may begin a *transition*, causing some of:

- The emission of signals using *output*.
- Operations on data, or other textual algorithmic code in a *task*.
- Alternate paths for the transition based on a *decision*.
- Exchange of remote variables through an *export* or *import* expression.
- Manipulation of timers using *set* or *reset*.
- A change of state – ending the transition.

Different input signals may trigger different transitions. If the current state does not have an input which will consume the first signal in the queue, then that signal is discarded. In the consumption of a signal at an input, the values carried in the signal parameters may be assigned to local variables. Figure 3.10 shows the basic symbols which are used to create an SDL state machine. A range of these symbols can be seen in action in Figure 3.11, which shows the SDL/GR description of the `receive_unit` process of the PTPMAC protocol description. Figure 3.12 further illustrates the use of SDL in a part of the PTPMAC transmit unit implementation.

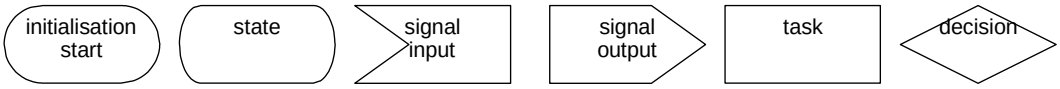


Figure 3.10: Basic SDL Symbols

SDL also includes the notion of *timers* which may be declared in a similar way to variables in finite state machines. Timers may be *set* to expire at a certain time (typically specified relative to the current time). An active timer may also be *reset* or cancelled. Upon expiry of a timer, a signal (with the same name as that timer) is placed in the input queue for the agent. This allows the modelling of real-time behaviour. The use of an SDL timer is illustrated in Figure 3.12 in the case of the lost token timeout timer: `tx_timeout`.

Tool Support

A range of tools is available to support the SDL design process. The SanDriLa add-on for Microsoft Office Visio [82], provides for graphical design capture, state analysis, and syntax checking. SDL/PR may be exported to allow integration with other tools. The Telelogic TAU SDL suite [83] provides a more thorough design environment (with correspondingly higher cost), including design capture, analysis, simulation, and automatic code-generation facilities. Cinderella SDL [84] provides similar features for graphical design capture, analysis and simulation of SDL specifications. Cinderella primarily supports the SDL-92 and SDL-96 variants, but also partially supports SDL-2000.

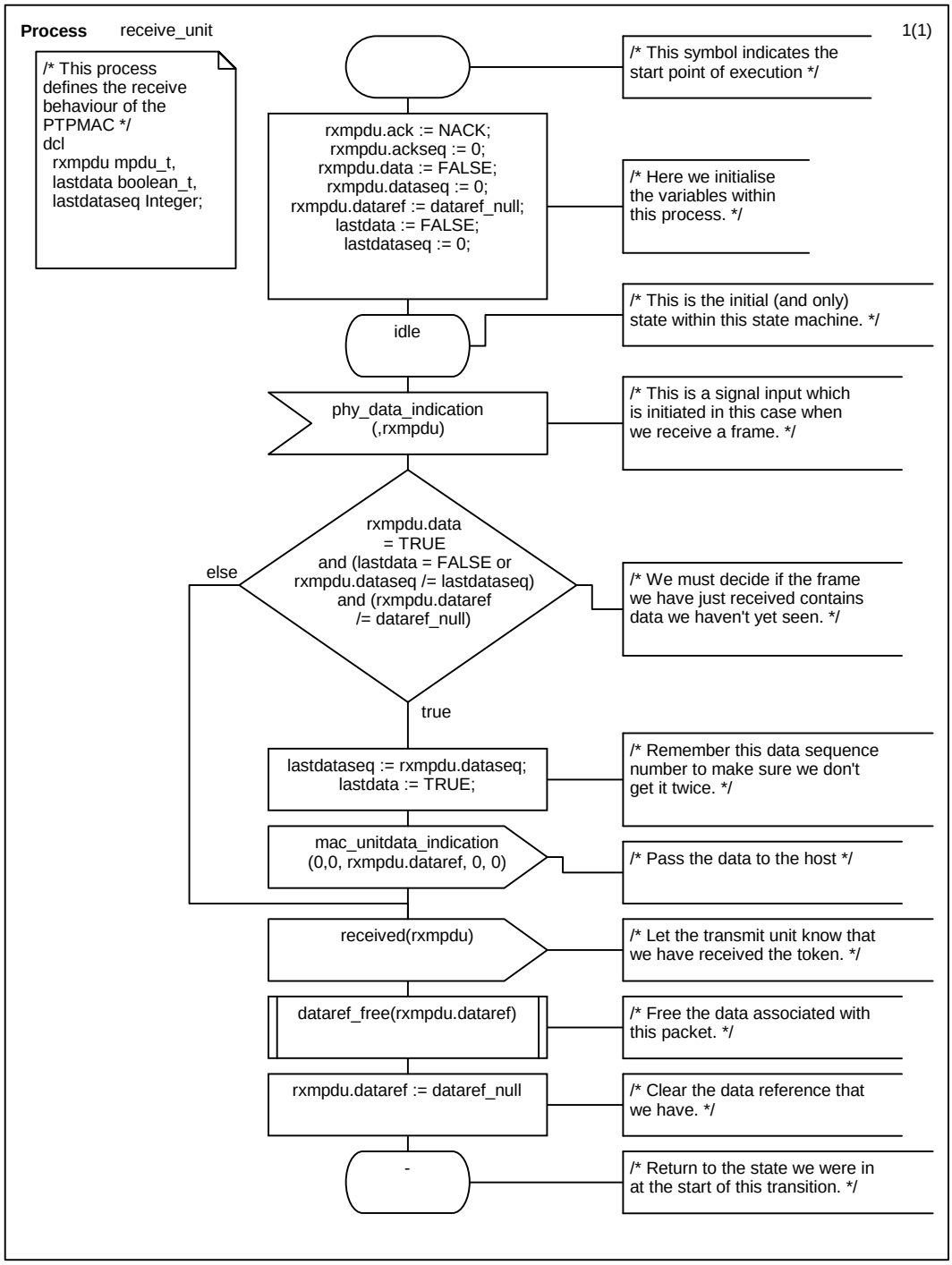


Figure 3.11: PTPMAC Receive Unit in SDL

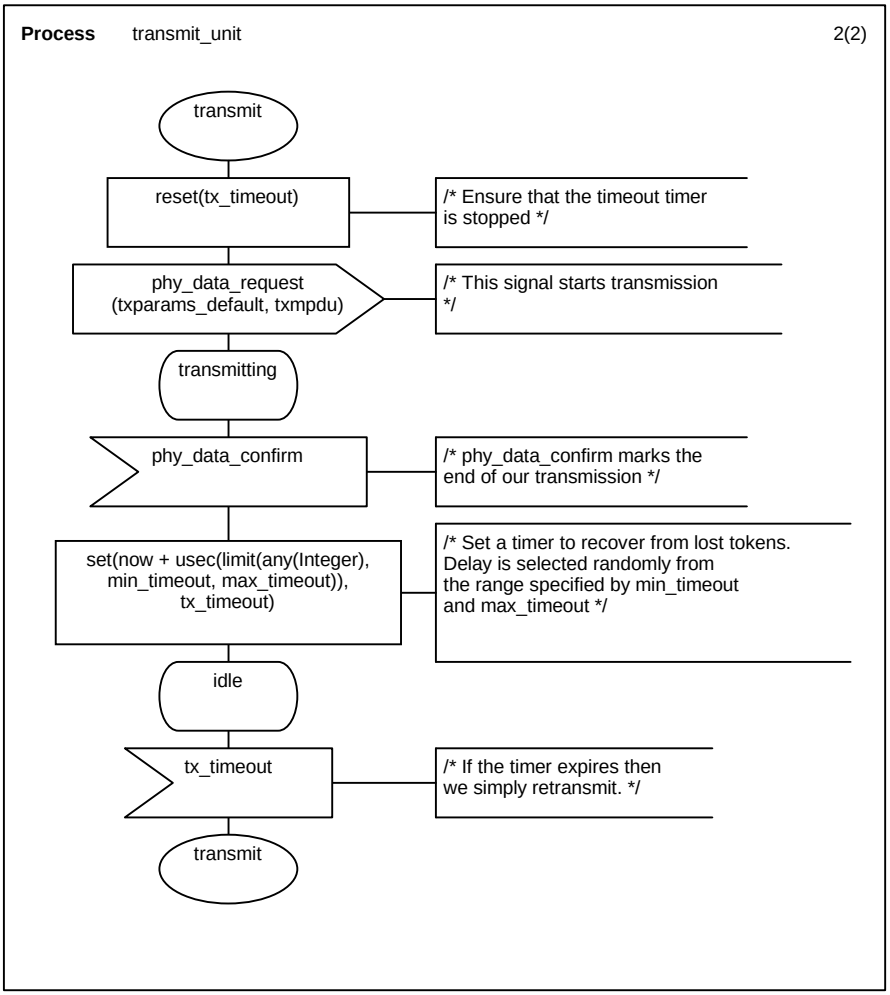


Figure 3.12: Partial PTPMAC Transmit Unit in SDL

3.3 Summary

The three formal description techniques discussed here have different strengths and weaknesses, and all have found application in protocol specification. E-LOTOS provides powerful structures for describing behaviour, however – as can be seen from the brief presentation of the language earlier in this chapter – it presents a significant learning curve to the uninitiated. Understanding large specifications can be a daunting task due to the lack of an associated graphical representation. E-LOTOS also suffers from a lack of industry-proven tools for design entry, verification and translation.

Being based on finite state machines, ESTELLE is generally intuitive for protocol engineers to use. The textual language is simple to grasp for those with experience of Pascal and similar imperative programming languages. ESTELLE/GR provides a solution to the need for a graphical representation, though tool support for this is limited due to the fact that it is not standardised.

SDL is a formal description technique which has been widely used in academia and industry. It has well-defined textual and graphical representations, which are both standardised and actively developed. Like ESTELLE, SDL provides a relatively simple path to formal description for those who have a basic understanding of finite state machines.

Given that each FDT presented has the capability for capturing the functionality that typically comprises a MAC protocol, two factors are of primary importance in assessing these formal description techniques: tool support; and standardisation. Of the three techniques discussed, SDL is the only one which rates strongly in these categories [13]. For these reasons, SDL has been chosen as the formal description technique for use in the development platform described within this thesis.

Chapter 4

Compilation

The target design platform aims to provide for the rapid development of MAC protocols. In order to do this, it must support: graphical design capture; use of library components for common functionality; hooks for integration into simulation environments at various levels of abstraction; automated synthesis of implementations and insertion of measurement test points. Given the choice of the Specification and Description Language (SDL) for the capture of design functionality, we must now begin to consider the remaining aspects of the framework.

The idealised model presented in Chapter 1 calls for evaluation-driven refinement of the initial specification toward an implementation, with evaluation methods including simulation, emulation, and measurement of prototype implementations. Simulation provides a means of verifying protocol correctness and evaluating performance at an early stage of the design cycle. It is extremely flexible in that model and system parameters can be readily altered to facilitate full exploration of the proposed design solution. Simulation of the SDL specification is a first step, with support from the Cinderella SDL tool introduced in Chapter 3 for design capture enabling evaluation at the earliest stages of protocol design. Accuracy of simulation traffic and channel models is vital however [17], and as the design cycle proceeds it is necessary to ensure that these are representative of the target application and environment.

Specialised network simulators such as *ns-2* [24, 85] provide models which allow accurate representation of network protocols. Such simulators typically provide a range of parameterisable traffic and channel models which can be tuned to match the specific characteristics of the system of interest. Such platforms sometimes also support emulation – the injection of live traffic into a simulated system. This approach can provide further confirmation of the validity of simulation results given the target application. To support effective evaluation of protocols within the Waikato Protocol Development Environment (WPDE) framework, it is necessary to allow integration into simulation environments such as that provided by the *ns-2* simulator. Beyond simulation, prototype implementation provides for evaluation in the actual target environment. This serves to verify the suitability of the protocol for its intended application, and validate any assumptions made in simulation models. The implementation platform described later in Chapter 6 is another environment into which we wish to integrate our MAC protocols.

The integration of protocols into these different environments will preferably be supported with minimal manual effort, in order to reduce the possibility of human error. Automated tools can provide support for this, ensuring consistency between representations at different design stages allowing direct comparison of evaluation results. Before considering such tools however, it is necessary to develop a standardised set of outer interfaces for MAC protocols developed within this framework.

4.1 Defining MAC Sub-Layer Interfaces

In the layered network model, each layer or sub-layer provides a service to its adjacent higher layer by using the services provided by the adjacent lower layer and building on them. Services can be defined in terms of the information exchanged between adjacent layers or sub-layers. This exchange of information is modelled by discrete, instantaneous events which may have zero or more parameters. These are known as *service primitives* [86].

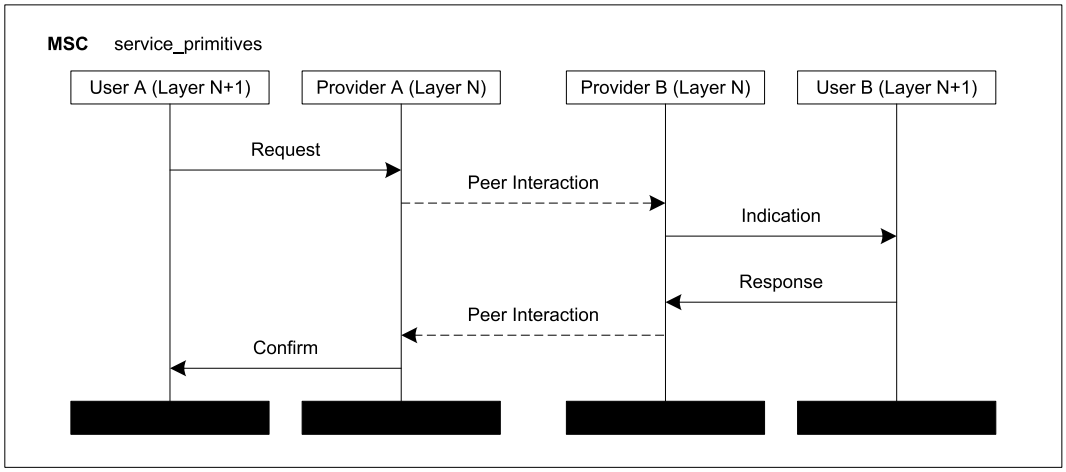


Figure 4.1: Types of Service Primitive

The convention followed within this work is based on that used by specifications within the IEEE 802 standards for local and metropolitan area networks. This approach classifies service primitives into four categories: A *request* is issued from a service user to a service provider to request that a service be initiated; a *confirm* is issued from a service provider to a service user to convey the results of a prior service request; an *indication* is issued from a service provider to a service user to notify it of an event within the provider which has significance to the user; and a *response* is issued from a service user to a service provider to complete a procedure initiated by the issue of an indication primitive. The MSC in Figure 4.1 illustrates this relationship.

Three primary classes of interaction occur between the typical MAC sub-layer and adjacent entities: a higher layer must interact with the MAC in order to use the fundamental data transfer service that it provides; the MAC must use the service provided by the physical layer to communicate with a peer MAC entity; some entity must provide control of MAC functionality in order to maintain its data transfer service to the higher layer. A further class of interaction is defined to support simulation and measurement during protocol development. This interaction allows internal MAC state to be reported to an external measurement entity. We now discuss these classes of interaction and define primitives for each.

4.1.1 MAC Data Service

The MAC sub-layer lies within the data-link layer as defined by the OSI layered model. The primary service provided by the MAC sub-layer to higher layers is the transfer of data between peer MAC entities. The Logical Link Control (LLC) sub-layer defined by the IEEE 802.2 standard [86], lies immediately above the MAC sub-layer within the data-link layer. The LLC sub-layer provides three types of service to the network layer: unacknowledged connectionless-mode services; acknowledged connectionless-mode services; and connection-mode services. These services are provided by building on those of the underlying MAC sub-layer. Various MAC layers are defined as standards by the IEEE, including 802.3 (Ethernet), 802.5 (Token Ring), and 802.11 (Wi-Fi). These seamlessly integrate with 802.2 due to the standardised MAC/LLC interface. There are clear benefits, in terms of interoperability, to generally following this interface within a MAC development framework.

In defining our primitives we will initially only provide support for the unacknowledged connectionless-mode services specified in the IEEE 802.2 LLC standard. This is the most common type of service used in practice, and generally supports the ubiquitous IP. To support this our MAC must allow delivery of units of data – each an MAC Service Data Unit (MSDU).

MAC-UNITDATA.request

We require a means by which the LLC layer can initiate the transfer of an MSDU to a peer LLC. In keeping with the IEEE conventions, this functionality is provided by the MAC-UNITDATA.request primitive. The issue of this primitive will generally cause the appropriate physical layer functions to be invoked to provide at least a reasonable attempt at successful delivery of the corresponding data unit.

The obvious parameters that are required for this type of request are the unit of data, and a destination address. Modern protocols are placing increasing emphasis on QoS, so a parameter allow specification of the MSDU priority is also included. Finally, a fourth parameter provides for MAC-specific flags.

The parameters of this primitive are shown in the table below.

Parameter	Description
destination_address	The address of the destination MAC sub-layer, or alternatively a group address.
data	The MSDU to be transferred.
priority	Desired priority for the MSDU.
flags	Field to allow protocol specific flags for the MSDU.

MAC-UNITDATA.confirm

Higher layers of the network stack can often generate data at a higher rate than the MAC sub-layer can deal with. This may be due to limited bandwidth available at the physical layer, or a performance bottleneck within the MAC sub-layer. To ensure that the MAC sub-layer is not swamped with data we require a handshake in the MSDU transfer, initiated by a MAC-UNITDATA.request. This is provided by the MAC-UNITDATA.confirm primitive. The single parameter to this primitive indicates the status of the prior MAC-UNITDATA.request, allowing the MAC to feedback status information to the higher layer.

Parameter	Description
status	The status or result code of given request.

MAC-UNITDATA.indication

When the MAC has an MSDU to pass to higher layer – generally because of a reception from a peer MAC – it must issue a primitive to the higher layer to convey this. This functionality is provided by the MAC-UNITDATA.indication primitive. The parameters associated with this primitive follow those specified for the MAC-UNITDATA.request, with the addition of the address of the source MAC entity.

Parameter	Description
source_address	The source MAC sub-layer entity address.
destination_address	The destination MAC sub-layer entity address, or alternatively a group address.
data	The MSDU that has been transferred.
priority	Desired priority for the MSDU.
flags	Protocol-specific flags associated with this MSDU.

4.1.2 Physical Layer Interface

The MSDU transfer service provided by the MAC protocol builds on the service available from the Physical Layer (PHY). The PHY interface provides for these interactions between the MAC layer and the hardware of the transceiver. The basic primitives support the transfer of data, as well as the exchange of information on the perceived channel state from the radio to the MAC, allowing the MAC to make informed decisions on when to transmit.

The PHY services required by the MAC are clearly dependent on the specifics of its function which is generally closely related to that of the underlying technology. Many parameters may be available for a given physical technology, including modulation type and rates, frequency of operation, transmit power. Physical layers supporting multiple modulation types are typically coupled with protocols which make use of this to provide features such as back-off to more robust modulation parameters in the presence of errors. In contrast, while a protocol such as Bluetooth requires the ability to change the channel of operation on a per-packet basis, this is typically a more static configuration parameter in many other MAC protocols. To maximise flexibility it is desirable to provide an interface which allows as many parameters as possible to be configured on a per-packet basis.

Similarly various parameters may be available on reception of a data unit, including the receive signal strength, and information regarding the modulation of the frame. This information is logically associated with each packet, and

so should be included with the notification of received data. Some MAC protocols require information on channel state, which may be based on detected in-band signal power, or acquisition of a carrier. This information is used in CSMA protocols as Clear Channel Assessment (CCA) to determine whether a transmission may begin or not. If a change in channel state is not communicated to the MAC in a timely fashion, then transmission collisions are likely to result.

The primitives described here are chosen to provide a basic but functional interface to the IEEE 802.11b-style physical layer which is used in the prototype implementation platform described later in Chapter 6. This physical layer provides a half-duplex (mutually exclusive transmit and receive) transceiver which encapsulates a MPDU with physical layer framing and broadcasts it on the medium in the transmit direction, and performs the reverse operation on receive.

PHY-DATA.request

To initiate transmission of an MPDU, the MAC issues the PHY-DATA.request primitive to the physical layer. The parameters for this request are the MPDU to transmit, and a set of transmission parameters which may include data rate and modulation options, and transmit power level.

Parameter	**Description**
txparams	Parameters for the physical layer.
data	The MPDU to transmit.

PHY-DATA.confirm

Flow control is required to avoid the MAC swamping the physical layer with data. This is provided by the PHY-DATA.confirm primitive which marks the completion of transmission of an MPDU previously requested using the PHY-DATA.confirm primitive. No parameters are associated with this primitive.

PHY-DATA.indication

When a frame is received the physical layer must pass it to the MAC layer. This is achieved using the PHY-DATA.indication primitive. The parameters of this primitive include the received MPDU, along with the set of reception parameters which may include receive signal strength, data rate, and timestamp.

Parameter	Description
rxparams	Parameters of the received MPDU.
data	The received MPDU.

PHY-STATUS.indication

CSMA protocols such as IEEE 802.11 require the ability to sense the channel prior to transmitting. To support such MAC protocols, the physical layer must provide a primitive to indicate a change in the perceived channel state. The PHY-STATUS.indication primitive provides for this. The sole parameter of this primitive indicates whether the channel is now considered busy or idle.

Parameter	Description
cs_state	The channel state as detected by the carrier sense mechanism – either idle or busy.

4.1.3 MAC Sub-Layer Management Entity Interface

Modern wireless MAC protocols do not generally operate in an autonomous fashion. Control is required for forming and breaking connections as well as managing the vast number of operational parameters that they have. In keeping with the terminology used in the IEEE 802.11 specification [2] we use the term MAC Sublayer Management Entity (MLME) to refer to the entity within the MAC boundaries which is responsible for the control required to allow the MAC to provide the data services that are its primary function.

The interface to the MLME has three main purposes: to allow basic control of the MAC sub-layer, including forming and breaking connections; to allow modification of MAC sub-layer parameters (e.g. timings or retry counts); and to allow management of advanced MAC sub-layer functionality such as QoS and security.

In practice, the MLME will typically interact – through the device driver – with a Station Management Entity (SME) such as Windows' Wireless Zero Config, or utility such as `iwtools` or `wpa_supplicant` in Linux. The functionality defined by the MLME is very much MAC protocol dependent. Procedures for forming and breaking connections, along with other operations such as bandwidth reservation, may be quite different for different styles of MAC.

A simple means of providing a generic interface to the MLME is through primitives allowing access to an undefined set of parameters or attributes. Any of the MAC-specific parameters may be virtual and have access side effects. This approach presents a concise, standard interface which is easily extended to suit a given protocol.

MLME-SET.request

The initiation of an MLME process is achieved in this model through a write to an MLME attribute. This is performed by the issue of the MLME-SET.request primitive. The parameters to this primitive are an identifier for the attribute which is being set, and the new value to be assigned to it.

Parameter	**Description**
attribute_id	The identifier for the MLME attribute.
value	The data element which is to be assigned to the specified MLME attribute.

MLME-GET.request

Correspondingly, it is necessary for the controlling entity to read back MLME attributes. This process is initiated by the issue of the MLME-GET.request

primitive, with the sole parameter specifying the identifier for the attribute of interest. The read value is returned in a following MLME-GET.confirm.

Parameter	Description
attribute_id	The identifier for the MLME attribute which is requested.

MLME-GET.confirm

This primitive returns the read value of an MLME attribute that was previously requested using an MLME-GET.request. The sole parameter specifies the attribute value.

Parameter	Description
value	A data element holding the value of the requested MLME attribute.

4.1.4 Measurement Interface

Our target simulation and implementation environments will allow tracking of MAC and PHY interactions, however, the internal state of the protocol under test will not necessarily be clear. By providing an interface which allows measurement points to be built into the protocol description, a clearer picture can be derived from simulation and implementation measurement results.

MEASUREMENT.indication

The MEASUREMENT.indication provides a portable means of tracking internal protocol operation in simulation and implementation. A protocol may emit this signal at key points to allow analysis of internal operation with reference to other visible interactions. This primitive has two parameters which allow unique identification of measurement points.

Parameter	Description
measurement_id	The identifier for this measurement point.
arg	The argument for the measurement point.

4.2 SDL Translation

The primitives described in Section 4.1 define the primary external interfaces of our MAC sub-layer. Within our framework, these primitives will correspond to SDL signals which are exchanged between the specification and its environment. To support the seamless design flow discussed in Chapter 1 it is necessary to provide a framework for integration of an SDL protocol specification into our target simulation and implementation environments.

There are several possible ways of providing for execution of the SDL specification within these environments. One approach is to provide a 'virtual machine' within each target framework which interprets the SDL model directly. A drawback of SDL interpretation is the inflated resource requirement placed on the target environment. This means that for complex specifications, performance may be severely restricted. Just-in-time compilation is one method of addressing the performance issues of the interpretation approach. This method, however, shifts a large amount of complexity into the environment. The preferred approach is translation of the SDL specification into a language which can be compiled to produce a native code for the target platform. With a suitable framework, the translated specification can be integrated into the target environment, providing an implementation which is as efficient as the translation process allows.

The obvious candidates for the translation target are the C and C++ languages due to the widespread availability of compilers for various architectures, and their use in one of our target platforms – *ns-2* (discussed in Chapter 5). The C++ language provides a translation target which is able to be efficiently compiled and mixed with existing C source in target frameworks. The object-oriented and hierarchical nature of SDL descriptions also maps well onto a C++ implementation. In a simulation context multiple instances of the pro-

tocol model will typically be operating concurrently, and therefore must be encapsulated appropriately to allow easy replication and independent execution. The C++ class provides a concise means of achieving this.

Several commercial tools are available for use in conversion of SDL to C and C++. The Telelogic TAU SDL suite [83] provides a rich design environment with code generation targeting both of these languages. Cinderella SDL [84] provides a low-cost SDL graphical design entry and simulation platform, and C and C++ code generator add-ons can be purchased. MAC protocols are typically implemented using a combination of hardware and embedded software. Resource availability means that the embedded software component must be relatively light-weight and efficient. Both Telelogic TAU and Cinderella provide code generation engines targeted at such systems.

Though these commercial packages provide many of the features necessary for code generation in the environments we are targeting, they also have a significant attached cost which can put them out of reach of many researchers. Being general purpose, these tools are often limited in terms of the optimisations they can make without knowledge of the design context [87]. The efficiency of the generated code both in terms of execution speed and memory footprint size can be increased if the nature of the target environment is taken into account. For these reasons the *sdl2cpp* tool has been designed and implemented as a part of the work described in this thesis. This tool converts an SDL/PR representation of a protocol into C++ code suitable for use in simulation and implementation. It is described in the remainder of this chapter, and is the first element of the WPDE.

4.2.1 Data-types

The mapping of basic SDL data-types onto our C++ target is relatively straight forward. Figure 4.2 shows mappings of structures, enumerated types, synonyms and syntypes onto C++. Syntactically, translation of these can be readily accomplished using relatively simple regular expressions; semantically, however, more consideration is required. As with the majority of abstract

```
value type mystruct;
 struct
  x, y, z Integer;
  b Boolean;
endvalue type;
```

```
struct mystruct {
  Integer x;
  Integer y;
  Integer z;
  Boolean b;
};
```

```
value type myenum;
literals
 one, two, three;
endvalue type;
```

sdl2cpp

```
enum myenum {
  one,
  two,
  three,
};
```

```
synonym pi Real = 3.141592654;
```

```
const Real pi = 3.141592654;
```

```
syntype address = Integer
endsyntype;
```

```
typedef address Integer;
```

Figure 4.2: Translation of SDL Data-types

modelling techniques, SDL assumes a world where processing is instantaneous, communication is error-free, and input signal queues have infinite length. Care must be taken to ensure consistent semantics between the formal specification and an implementation where only finite resources are available [88].

Data-types provide a similar complication for the mapping of formal descriptions onto concrete implementations. Data-types in languages such as SDL are abstract and infinite. Integers in SDL, for example, may take on any value. In an implementation, variables are bound to lie within the range permitted by the number of bits in their machine representation. A 32-bit integer is common on many C/C++ based systems, giving a maximum range of $-2,147,483,648$ to $+2,147,483,647$. While this range is sufficiently large for a majority of purposes, the semantic difference between the data-type in specification and implementation can counter some of the benefits of using formal techniques at the initial stages of design.

In the given example of the integer data-types it is not possible to match the semantics of the SDL version in the implementation. Though virtual data-types can be implemented with larger ranges, the implementation platform has finite resources and these will eventually be reached. An alternative is to match the semantics of the C/C++ data-types in the SDL model. The `syntype` in SDL allows this to be achieved:

```
syntype uint8_t = integer constants 0:255; endsyntype;
```

It is also important to consider the relationship between the more complex SDL data-types that may be used and their C/C++ implementations. A primary function of any MAC protocol is the transfer of data between the physical and host interfaces. This has the potential to be a processor intensive aspect of operation in an implementation if care is not taken to reduce the amount of data copy that is required. Copy avoidance is therefore a key optimisation for network protocol implementations [89].

The predefined SDL Octetstring data-type is commonly used as a representation for a block of data. This type is defined as an unbounded list of bit-strings of size 8. Concatenation and extraction operators are defined to allow creation and manipulation of these data-types. Though similar data-types could be emulated in an implementation, the semantics of SDL require that the transfer of an Octetstring as the parameter of an exchanged signal involve a replication of the data. As discussed above, this is undesirable in most MAC implementations.

An implementation in a general-purpose programming language such as C or C++ would avoid copy by communicating a pointer to the data structure concerned. Though the SDL language has no notion of pointers or references to data, it is possible to define an abstract type which encapsulates the typical functionality required to manipulate data. Operations on objects of this data-type within an SDL description can be translated to their implementation-specific versions to ensure efficiency.

The *data reference* is the abstraction defined within the various sections of the WPDE framework to allow manipulation of data units with minimal copy. Though the form and implementation of a data reference is unique to each platform, a common interface is provided by several operators to allow manipulation of these and the data they encapsulate. The data reference type (`dataref_t`) is central to the MAC development framework described - it is the method by which data will be exchanged with the host, and also the unit which should be passed as the PDU to the physical layer in an implementation.

By defining an API which allows for manipulation of data references, specific implementation or simulation platforms can provide an optimised version of the necessary functionality. The API must support creation and destruction of data references, along with insertion, manipulation, and extraction of the data within.

The data reference API is designed to be simple to use, and sufficiently abstract to allow implementation within the SDL and C++ environments in which it is required. Creation of data references is performed through the `dataref_alloc` and `dataref_alloc_size` operators, with the former creating and returning a data reference of maximum size, and the latter allowing the size to be constrained to optimise resource usage. Correspondingly, the `dataref_free` routine is called when a data reference is no longer required to free the resources associated with it.

A common requirement in MAC protocols is the encapsulation of an MSDU inside a MPDU, and the corresponding decapsulation at the receiving end. These can be achieved using the `dataref_cat` and `dataref_extract` calls. The first routine concatenates the contents of a source data reference onto a destination. The second will create a new data reference by extracting data from a specified source data reference, along with the offset and length specified in octets.

4.2.2 Communication and Scheduling

The SDL communication model is based on the asynchronous exchange of signals. A signal which is destined for an agent is placed in an infinite-length input queue from which it can be drawn at some later time. Clearly a real-world implementation can not have infinite resources so some approach must be taken to modelling the intended SDL behaviour.

One method is to make use of finite length signal buffers with overflow avoided through back-pressure implemented using a semaphore or other means. Sending processes will add signals to a buffer until that buffer becomes full at which

point the sending process will be temporarily suspended. This approach is known as the *server model* [87], and an example is described by Dietterle et al. [90] in their proposed mapping of an SDL description onto a TinyOS platform. Their described process maps SDL agents onto TinyOS components, with communication achieved using either command or event handlers at the designer's discretion. The server model has the advantage of maintaining a separation between process activation and communication. A significant drawback is the overhead both in terms of runtime environment complexity, and memory footprint that is required to implement this method.

An alternative approach is the modelling of SDL communication as an *activity thread* [87, 90]. Using the activity thread model, execution follows the transitions caused by a single external stimulus. This stimulus may be the injection of a signal from the environment into the system, or the expiry of a timer set within the system. Each signal exchange is modelled by the invocation by the sending agent of a subroutine associated with the receiving agent. This method of modelling an SDL system removes the need for a process scheduler, context switches, and signal queues, and therefore simplifies and optimises the generated code.

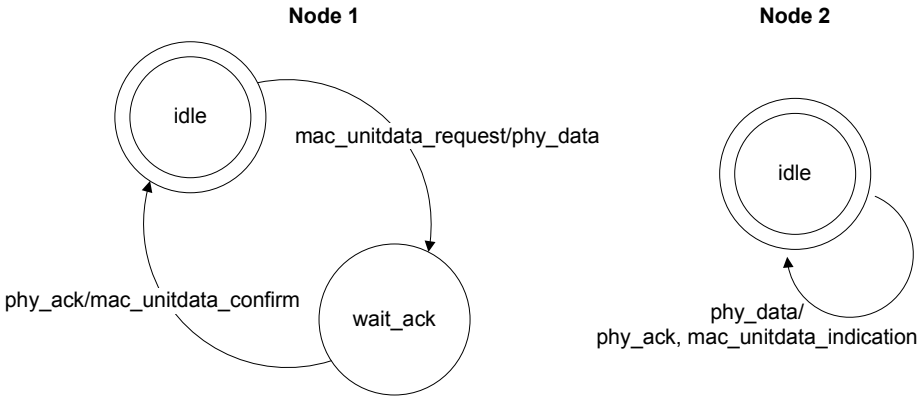
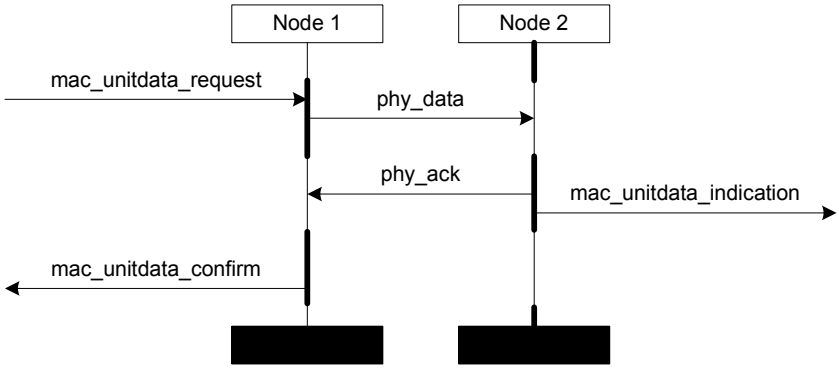
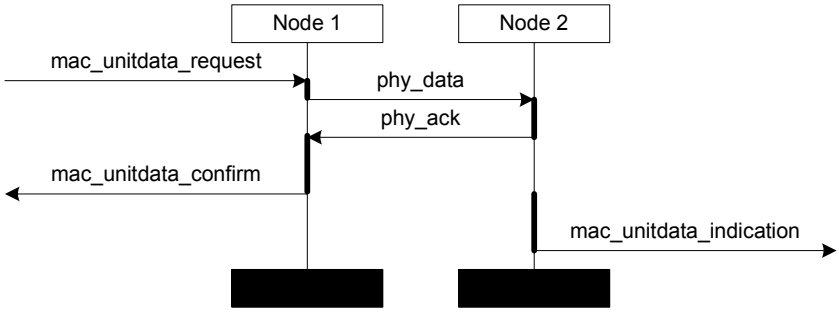


Figure 4.3: Communicating Finite-State Machines

Consider the two simple communicating state machines illustrated in Figure 4.3. Figures 4.4(a) and 4.4(b) illustrate the execution of a simple signal exchange in this system using the server and activity thread models respectively. The signal exchange is initiated by the receipt from the environment of an `mac_unitdata_request` signal. The path of execution is shown within the



(a) Server Model



(b) Activity Thread Model

Figure 4.4: SDL Execution in the Server and Activity Thread Models

message sequence charts by the solid line running down the vertical axes. In the server model depicted by Figure 4.4(a), execution of the two nodes is scheduled alternately, whereas in Figure 4.4(b) execution follows signal exchange.

Two key conditions must be satisfied in order for the activity thread approach to be valid: the communication sequence must take the form of a procedure call tree [88]; and signals must not arrive from the environment faster than the execution of a call tree allows. The latter of these conditions can be enforced through careful design of the runtime environment, but the former requires care to be taken in construction of the SDL model. Though this restricts the domain of SDL descriptions which can be correctly translated, it does not restrict the ability to model any functionality. Use of the activity thread approach provides significant complexity and efficiency benefits for implementations, and is sufficient to demonstrate the merits of the framework proposed in this thesis.

In order to implement the activity thread model in our C++ environments, SDL signals are represented as C structures and references to these are passed to ensure efficient memory usage. The `signal_t` structure defines a common header to allow identification of signals. This structure consists of a single signal identifier which is a platform-specific scalar type. Specific signal definitions contain the signal identifier as a first member before the parameters are specified in order. Parameters are named using the sequence `param`*n*. The code below shows the generic `signal_t` structure definition along with a specific signal implementation:

```
typedef struct signal_t {
  signalid_t    id;
} signal_t;

typedef struct mlme_get_request_t {
  signalid_t    id;
  Integer       param0;
} mlme_get_request_t;
```

4.2.3 Agents

As discussed in Chapter 3, an SDL description is made up of a structure of communicating agents. Each agent may contain variables, procedures, a state machine, and may encapsulate other agents. Clear parallels can be seen between the SDL agent, and the C++ class. By mapping each agent type to a C++ class we can readily provide these features in a tidy manner. Shown below is the abstract base class – `wpdeagent` – which provides the basis for specific agents.

```
class wpdeagent {
 public:
  wpdeagent(wpdeagent *p, wpdeagent *top);
  virtual void reset() = 0;
```

```
  virtual void input(wpdeagent *from, signal_t *sig) = 0;
 protected:
  wpdeagent *parent;
  wpdeagent *toplevel;
};
```

To implement the activity thread model of SDL communication the `wpdeagent` class includes a pure virtual method – `input()` – which is called to deliver each SDL signal. An invocation of this method must include parameters specifying the source agent and the signal being conveyed. The details of the `signal_t` type are discussed later. A class derived from `wpdeagent` for a particular agent or agent type will provide an implementation of the `input()` method with an invocation causing update to the encapsulated state machine, or relay of the signal to a child, or the parent agent. The parent agent is tracked through a protected member variable, and is set by the constructor on creation.

A similar virtual method – `reset()` – provides a means of reverting the agent and any children to their initial states. The agent-specific implementation of this method will perform the operations specified in the start transition of that agent (including the implicit initialisation of any variables), and then invoke the `reset()` method of any children.

Variables and child agents are defined by instantiation of the appropriate C++ class or data-type within the class of the encapsulating agent. The top-level agent class maintains copies of any exported SDL variables from agents within the system. Each agent holds a pointer to the top-level agent, and uses this to access these exported values. The SDL `export` operation involves a copy from the local variable instance to the exported instance in the top-level agent. Similarly, an `import` de-references the top-level pointer to access the global copy. The class definition of an example agent is shown in the code below. This agent encapsulates two child agents, a simple state machine, and a single integer variable:

```
class my_agent : public wpdeagent {
 public:
```

```
  my_agent(wpdeagent *p, wpdeagent *t);
  void reset();
  void input(wpdeagent *from, signal_t *sig);
 protected:
  // Child agents
  my_child_t  my_child;
  my_other_child_t  my_other_child;

  // States
  enum { state_1, state_2 } state;

  // Variables
  int my_count;
};
```

Shown below is the translation of the `input()` method for the example agent introduced above. Two sections are apparent in the method body: the signal routing section, and the state machine implementation. The signal routing section matches input signals based on their source and signal identifier, and forwards them to the appropriate destination. This is derived from the channel structure defined in the SDL description. The state machine section is similar, but matches include the current state, and leads to execution of the corresponding transition body:

```
void my_agent::input(wpdeagent *from, signal_t *sig)
{
  // Routing
  if (from == &my_child && sig->id == sig_to_env_id) {
    parent->input(this, sig);
    return;
  }
  if (from == parent && sig->id == sig_to_child_id) {
    my_child.input(this, sig);
    return;
```

```
  }
  if (from == &my_child && sig->id == sig_to_other_child_id) {
    my_other_child.input(this, sig);
    return;
  }

  // State machine
  if (state == state_1) {
    if (sig->id == prod_id) {
      my_count = my_count + 1;
      state = state_2;
      return;
    }
  }
  if (state == state_2) {
    if (sig->id == prod_id) {
      my_count = my_count + 1;
      state = state_1;
      return;
    }
  }
}
```

4.2.4 Transitions

As discussed in Chapter 3, transitions in SDL contain the predominant functionality of the specification. Transition bodies may comprise numerous types of operation, including manipulation of data, output of signals, and state changes. The mapping of these into our activity thread C++ model is the only aspect of the translation process left to discuss. Figure 4.5 illustrates a selection of SDL constructs and their C++ translations.

Expressions have similar forms in both languages, though some operators have slightly different forms requiring simple conversion as in the case of the assign-

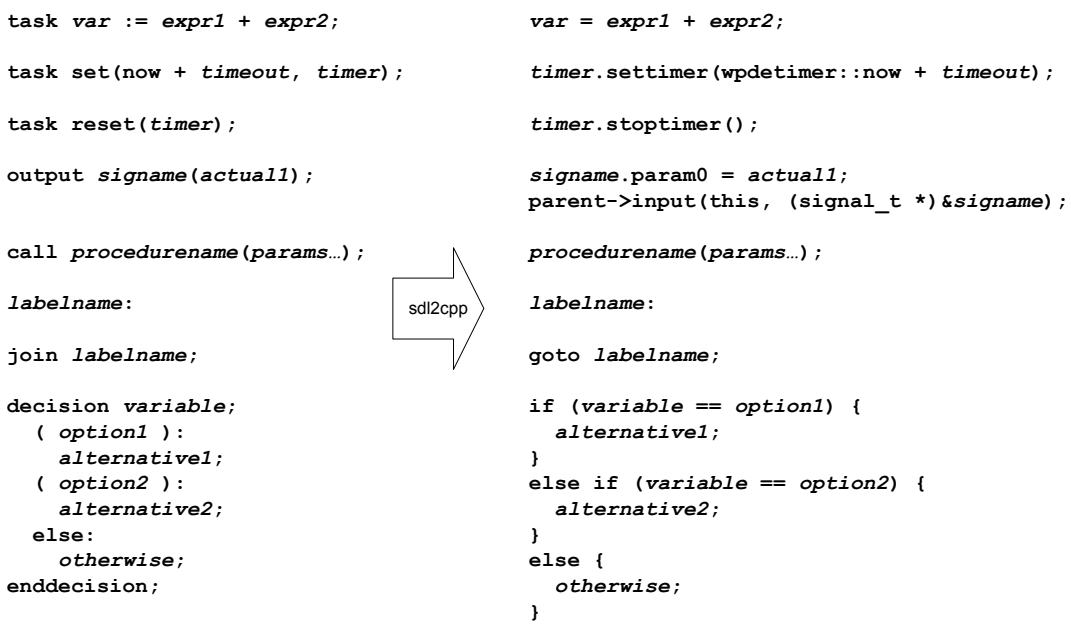


Figure 4.5: Translation of SDL Transitions

ment operator conversion from '`:=`' to '`=`'. Tasks involving timer manipulation map to invocation of routines of the timer API defined earlier. Signal exchange is modelled by the passing of references to our signal structures. Each agent class holds an instance of every possible signal that it may output. The output of a signal translates into the assignment of the actual expressions to each of the signal structure parameter members, along with the invocation of the `input()` method of the parent agent. The signal identifiers within each structure are set within the class constructor. Procedure calls, labels and join statements are trivially converted to their C++ equivalents, as is the conditional execution provided by the SDL `decision`.

4.2.5 Compiler Structure

The operation of the SDL-to-C++ compiler is illustrated in Figure 4.6. An SDL-2000 parser written by Michael Schmitt [91] is used for the initial stages of lexical analysis and token parsing. This open-source parser uses the *ANTLR* compiler construction tool for generation of the lexer and parser, and provides a framework for implementation of an Abstract Syntax Tree (AST) parser.

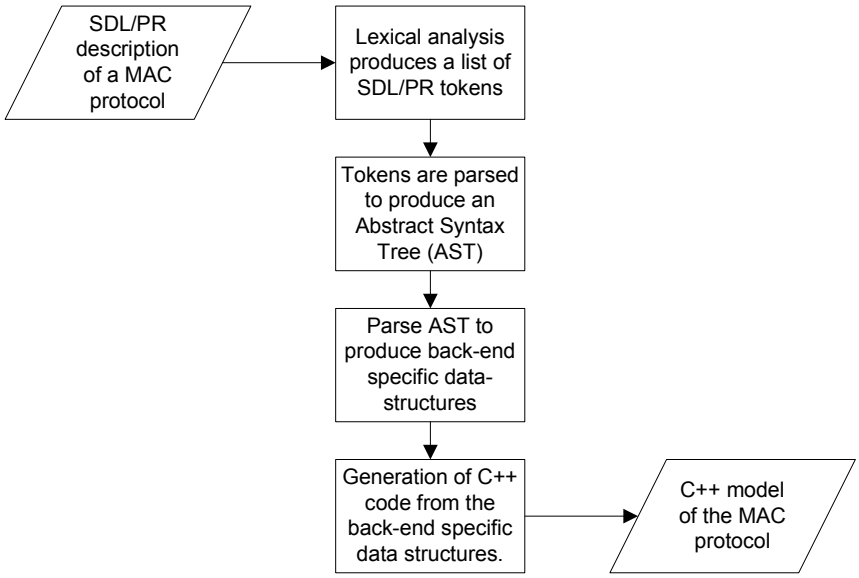


Figure 4.6: SDL to C++ Translation Process

The AST parse and code generation phases are implemented as C++ integrated with the *ANTLR* front-end. The AST parse phase involves walking over the AST and generating data structures more in-line with the target code. At this stage, sub-blocks of code are generated for transitions, and variable and agent instantiations are collated ready for the final stage. The final phase is the code generation from the data structures created in the previous step. For each SDL agent three files are created – a types header, the agent class header, and the class implementation file. The types header contains definitions for any SDL types which were defined within the agent. It also provides the unique signal identifiers and layouts for any declared signals.

The class header declares the agent object which has a base class of `wpdeagent`. Within this class are defined any variables and timers local to that agent, any contained agents, and an instance of each signal type that can be output by that agent. For the top-level agent, member variables include shadow copies for all exported variables in the system. Prototypes for PDU encapsulation and decapsulation routines are also included in the top-level agent class.

The class implementation file contains the body of the three primary agent routines: the class constructor, the `reset()` method, and the `input()` method.

The constructor initialises any contained agents and also sets up the unique signal identifiers for signals within the output signal set. The `reset()` method contains the code generated for the start transition, along with invocation of the corresponding methods of any contained agents. Within the `input()` method the signal routing and state machine code is generated.

4.3 Summary

The service definitions and translation strategy described in this chapter provide the framework required to automatically convert an SDL specification of a MAC protocol into a C++ model suitable for integration into network simulators like *ns-2*, and implementation environments such as that later described in Chapter 6. Key advantages of the strategy proposed here are the lightweight nature of the code generated. Despite the relative simplicity of the approach, sufficiently complex specifications can be translated.

Chapter 5

Simulation

The target design flow discussed in Chapters 1 and 2 involves a iterative process of refinement from initial specification through to a prototype implementation. Refinements are driven by repeated evaluation of the protocol in terms of correctness and performance. Simulation provides an evaluation tool which can be used at a stage in the development cycle before sufficient detail has been introduced into the design to support an implementation. This is not to say that simulation only has a use in the initial stages of protocol development. Quite to the contrary, simulation allows careful control of environment and stimulus and ready access to the internals of the system under test. These characteristics make it a valuable evaluation tool at all stages of the protocol design lifetime if appropriate rigour is applied [23].

Kotz et al. [17] provide several recommendations for the wireless network research community in terms of improving the usefulness of simulation techniques. A central theme in these is the importance of using appropriate models within a simulation environment. This applies not only to simulation traffic models, but also to the representation of the underlying physical layer. A second recommendation which has significance here is the need to allow the use of the same code within simulation and implementation environments. This allows direct comparison of the results of simulation and implementation evaluation, and provides a means of verifying or refining the simulation models.

Finding the appropriate level of model detail for use in performance simulation is an important but challenging task [17, 51]. Increased levels of detail can significantly increase the run-time of simulations, particularly in scenarios such as those associated with large-scale sensor networks, which may comprise thousands of nodes [92]. The trade-off associated with model abstraction, however, is potential for incorrect or irrelevant simulation results. This is particularly applicable to the lower layers of the network stack. Differing characteristics of such layers can impact severely on network and transport layer performance. Correspondingly, the simulation performance hit of excessive model detail is magnified in these layers due to the tendency for their functions to be invoked multiple times for a single higher layer operation.

The allowable level of abstraction within simulation models varies throughout the design lifetime. For the purpose of verifying correct protocol operation, manual stimulus will generally suffice for a traffic model to allow inspection of the basic frame exchange sequences. Similarly, a model of a reliable channel is often all that is required to approximate the physical layer. At later stages of evaluation, more detailed traffic and channel models must be used to provide representative indications of protocol performance.

Identifying the relevant performance metrics for a given protocol or study is a key step toward selecting acceptable simulation model abstractions [18]. Performance requirements are generally set with reference to the application layer, and may include specifications for power efficiency, throughput, latency introduced by the MAC sub-layer, ability to provide QoS differentiation, or any number of other possibilities. Acceptable model abstractions are generally determined experimentally through comparison of the relevant metrics given by corresponding simulation and implementation studies. In the development of a new MAC protocol this approach is not possible until well into the design process, and so it is necessary to start with a high level of detail within the MAC simulation model. Similarly, it is important to ensure that physical layer and traffic models are as representative as possible of the target application environment [17]. Given the idealised design flow in Chapter 1, the use of a common design representation through the development lifetime ensures that all available detail is present within the MAC simulation model. Suitable

physical layer and traffic models must be provided for by the performance simulation environment.

Two distinct environments cater for simulation in the framework described. Direct simulation of the protocol specification within the Cinderella SDL [84] tool provides for early analysis of protocol operation and verification of the correctness of frame exchanges and state machine interactions. Performance simulation is catered for through integration into the *ns-2* network simulator which provides good models for application traffic generation, transport and network layer protocols, and the wireless channel. In this chapter we discuss how these simulators are integrated into the presented design flow. Though careful simulation using appropriate models can provide useful and early evaluation of a protocol under development, the risk of model inaccuracies affecting results can be removed through actual implementation of the protocol. A platform for prototype implementation of protocols within the described framework is presented later in Chapter 6.

5.1 SDL Simulation

The use of simulation at early stages of the development process accelerates the initial design cycle and allows the design space to be thoroughly evaluated before the developer becomes immersed in low-level design detail. Given the choice of SDL as an initial representation, there is an opportunity for simulation of the specification in its raw form. Various SDL simulators exist, and these are often coupled with graphical SDL design environments – the Cinderella SDL tool introduced in Chapter 3 is an example. At this simulation level we have three goals: to determine whether the protocol we have specified works; to detect any differences between the description we have produced and any prior informal specification or vision of how the protocol should operate; and to identify any inefficiencies which are apparent at this level of simulation detail.

The boundary of our simulation is defined by what is known as a testbench –

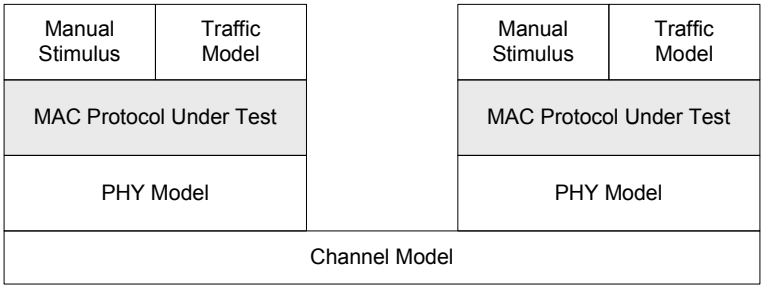


Figure 5.1: SDL Simulation Elements

the virtual environment which allows evaluation of the device (or in this case, protocol) under test. Generally the testbench should be self-contained, thus requiring little or no external stimulus in operation. Figure 5.1 illustrates the elements of a testbench for use within the SDL simulation environment. This testbench allows modelling of the lower layers of the network stack. Stimulus is provided directly to the MAC sub-layer by traffic models or manually by the user. Each MAC sub-layer has an associated physical layer model which represents the corresponding wireless transceiver, and these are coupled through the closely related channel model. Given the goals of this simulation level, traffic models may be simplistic involving independent traffic sources and sinks. More detailed models may incorporate both aspects to allow dynamic reaction to received traffic.

Though specialised network simulation platforms such as will be used for performance simulation later in the design cycle incorporate traffic and channel models, SDL simulation environments are typically general purpose and lack these specific components. The SDL simulation environment in Cinderella provides execution of the specification, and generates a trace of signal exchange which can be viewed within the Graphical User Interface (GUI) as a MSC. Signals can be manually inserted into the input queue of any process instance within the system, and features similar to those found in most debuggers allow viewing and manipulation of process state including local variables, and the tracing of execution with breakpoints and stepping facilities. To support the simulations desired within our framework it is necessary to implement traffic and physical layer simulation models in SDL. We consider first the modelling of the physical layer of our simulated wireless network.

5.1.1 Modelling the Physical Layer

Figure 5.2 illustrates a primitive model of a shared medium. The channel can be considered to be in any one of three states: idle, busy, or collision. The channel is considered to be idle when no transmitter is currently active. If a transmission begins, then the channel is considered busy, and if the completion of that transmission is the next event, then the data is successfully transferred, and the channel becomes idle again. In this simple model, the busy state represents the period when data is being transferred to any listening stations. If another transmission begins while the channel is busy, then the model dictates that the channel move to the collision state, indicating the corruption of data. The channel remains in the collision state until such time as no transmitters are active, at which point it becomes idle, with no transfer of data.

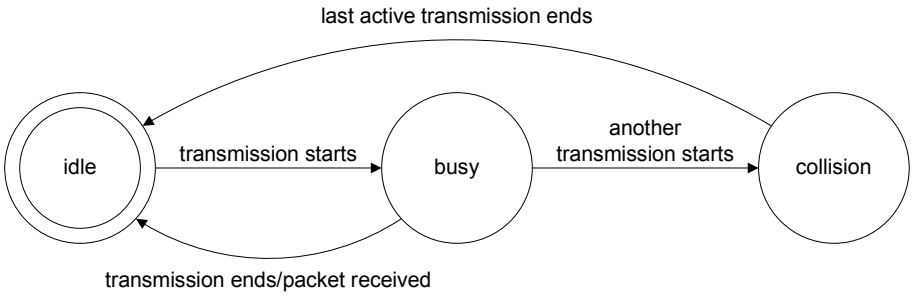


Figure 5.2: Modelling a Shared Medium

Typical wireless receivers are able to track and potentially receive without error the stronger of two signals, if the relative difference in power is greater than a factor known as the *capture ratio* [93]. This behaviour is known as the *capture effect*. The model of Figure 5.2 provides a good approximation for use in channels such as Ethernet. In a wireless environment however, the capture effect invalidates the assumption that temporally overlapping transmissions will both be lost. By adding the concept of received signal strength we can incorporate the capture effect into our wireless channel. This requires that we take into account the two major contributors to the received signal strength – transmitter power and the physical distance between transmitter and receiver[1]. This added detail reflects the fact that different wireless receivers will have

[1]Random variation in the received signal power level is also possible due to environmental conditions, but at this stage we omit this detail from our model

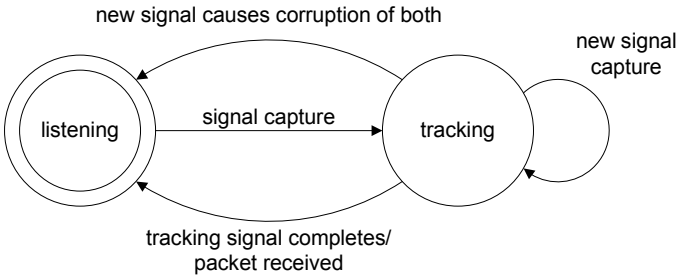


Figure 5.3: Model of a Channel with the Capture Effect

different views of the channel. For this reason we change our model from an overall channel state, to a node specific receiver state. Figure 5.3 shows the improved channel model.

Two receiver states are specified in this model. In the *listening* state, the receiver does not have carrier lock on any transmitted signal. This may be because no transmitter is active, or simply because no signal is sufficiently strong to capture the receiver. When a signal is sufficiently strong to capture the receiver, the receiver will begin *tracking* that signal. If this situation remains until completion of the transmission which is being tracked, then the data is considered to be successfully received. If a new signal becomes present, and is stronger than the original signal by more than the capture ratio, then the receiver ceases its tracking of the original and begins tracking the new signal. The model reflects this by remaining in the *tracking* state. If the new signal is strong enough to interfere with the original signal and prevent the continued tracking of that signal, but is not strong enough to capture the receiver, then a collision occurs and neither signal results in a received frame. In this case the model returns to the *listening* state.

In the models discussed so far, delivery of frames has been all-or-nothing. If a frame is affected by collision, or otherwise corrupted with respect to a particular node, then that node is deemed not to have received the frame. However, this does not accurately reflect the behaviour exhibited by real wireless devices. The all-or-nothing approach to modelling error may be sufficient for some protocols. The 802.11 protocol, for example, makes use of a 32-bit CRC to cover the entire frame body – thus, any error will cause the frame to be dropped. In contrast, a Bluetooth SCO frame provides error checking only on

Feature	Required Detail
Reliable and instantaneous signal exchange	No extra information required
Collision for temporally overlapping channels	Frame duration on the physical layer: derived from physical layout of frame information
Modelling of receiver sensitivity and the capture effect	Node locations and transmit power levels
Realistic error models	Non-determinism applied to error based on received signal-to-noise ratio. Recording of error locations in frame
Power consumption profiling	Tracking of MAC/radio state, with power consumption models for the target implementation

Table 5.1: Levels of detail in the physical layer

its header, leaving the body of the frame unprotected. While any bit error in the header will cause the frame to be dropped within the Bluetooth baseband sub-layer, a frame with corrupted payload is still used. Clearly in simulating this protocol, it is necessary to include more detail than in our basic error model.

The detail we require in this case is more information on the nature of the errors. Depending on the amount of headroom between the tracked signal strength and the next strongest interferer (which may be ambient noise), the degree of error introduced to the received frame (if received at all) can vary from zero to total. If the margin is near the capture ratio, then the previously mentioned random variations in the received signal strength can cause both the bursty errors typically known as fading, and random bit errors. If the margin is smaller than the capture ratio, then bursts of corrupted data corresponding to the interfering activity can occur.

Another example of an instance in which it is important to match the level of detail to the simulation goals is in the study of power-aware protocols. For protocols which need to be power-aware it may be important to distinguish carefully between the states of the radio. A simple distinction between *listening*, and *receiving* may not be sufficient for the architecture in question,

as it may involve sub-states of these which have significantly different current consumption characteristics. Table 5.1 shows a summary of the physical layer behaviours discussed in this section, and the detail required to model at each stage.

We have not yet discussed the modelling of transmit behaviour. Fortunately this is relatively straightforward due to the modelling of competing signals at the receive end. Figure 5.4 illustrates the addition of transmit to our physical layer model. An additional *transmitting* state represents an active transmitter. The start of transmission invalidates any in-progress receptions. The physical layer model which we will use in our initial SDL simulations combines aspects of our initial basic model shown in Figure 5.2, with the transmit elements shown in Figure 5.4. The SDL/GR description of the physical layer is shown in Figure 5.5.

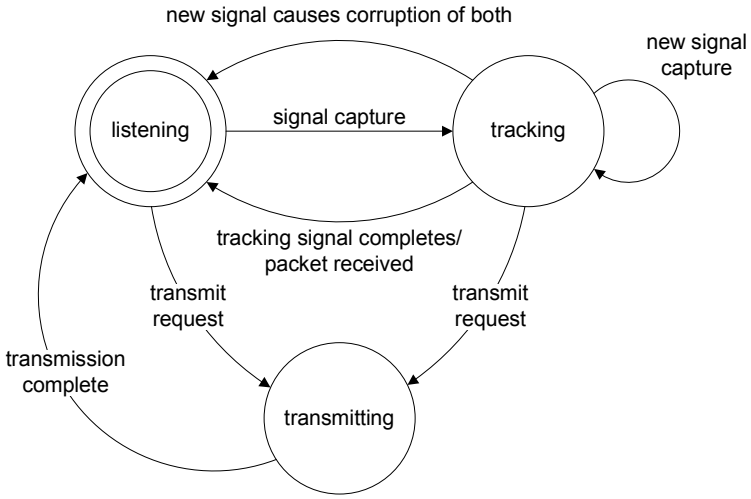


Figure 5.4: Modelling Transmit

Four states are maintained in the physical layer model: `idle`, `busy`, `collision`, and `transmitting`. Any temporal overlap on receive is considered to cause loss of all packets involved. Similarly, a reception which is interrupted by a local transmission will be considered lost. The network allocation vector (NAV) consisting of a timer – `navtimer` and a variable of type `time` – `nav`, tracks any activity on the medium. Expiry of the timer in the `busy` state represents the completion of a successful packet reception. In any other state, expiry of the timer represents the medium becoming idle with the end of the

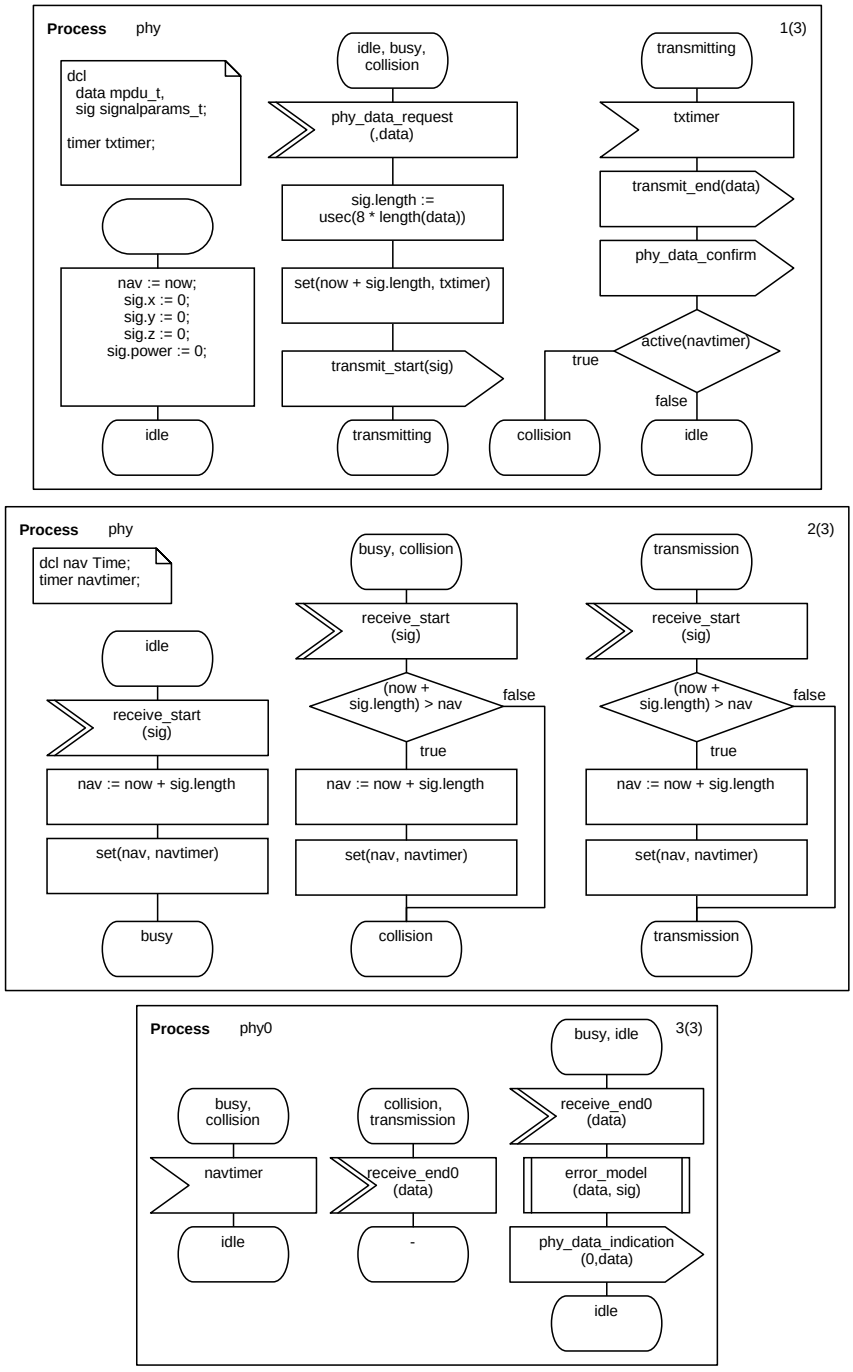


Figure 5.5: SDL Model of the Physical Layer

last outstanding remote transmission after a collision.

Packet errors may be modelled within the `error_model` procedure which is called by the physical layer model prior to a `phy_data_indication` signal being sent. The call to this procedure is visible in the third sheet of the model shown in Figure 5.5, and takes the MPDU and signal parameters as arguments, with the MPDU able to be modified to reflect error.

Communication on the simulated medium is by way of the `transmit_start`, `transmit_end`, `receive_start` and `receive_end` signals, with the former two issued by the originating node, and translated into the latter two by the medium model. The `transmit_start` and `receive_start` signals carry a parameter which defines the signal characteristics. The type of this parameter is defined as:

```
value type signalparams_t;
 struct
  length  Duration;
  x, y, z Real;
  power   Real;
endvalue type;
```

The `length` parameter indicates the duration of the transmission. The parameters `x`, `y`, and `z` indicate the location of the signal transmitter in space. These are unused in the current model, but provide for future extension, along with the final parameter – `power` – which indicates the transmitted signal power, and when coupled with transmitter and receiver locations, allows for received signal strength to be modelled. The `transmit_end` and `receive_end` signals serve to deliver the MPDU, and carry this as their sole parameter.

These signals are defined within the block shown in Figure 5.6, which instantiates two physical layer blocks in order to provide a simulation model for a point-to-point wireless channel. A single SDL process is necessary to avoid potential race conditions in the relay of the `transmit_start`, `transmit_end`, `receive_start` and `receive_end` signals introduced above. This can be seen

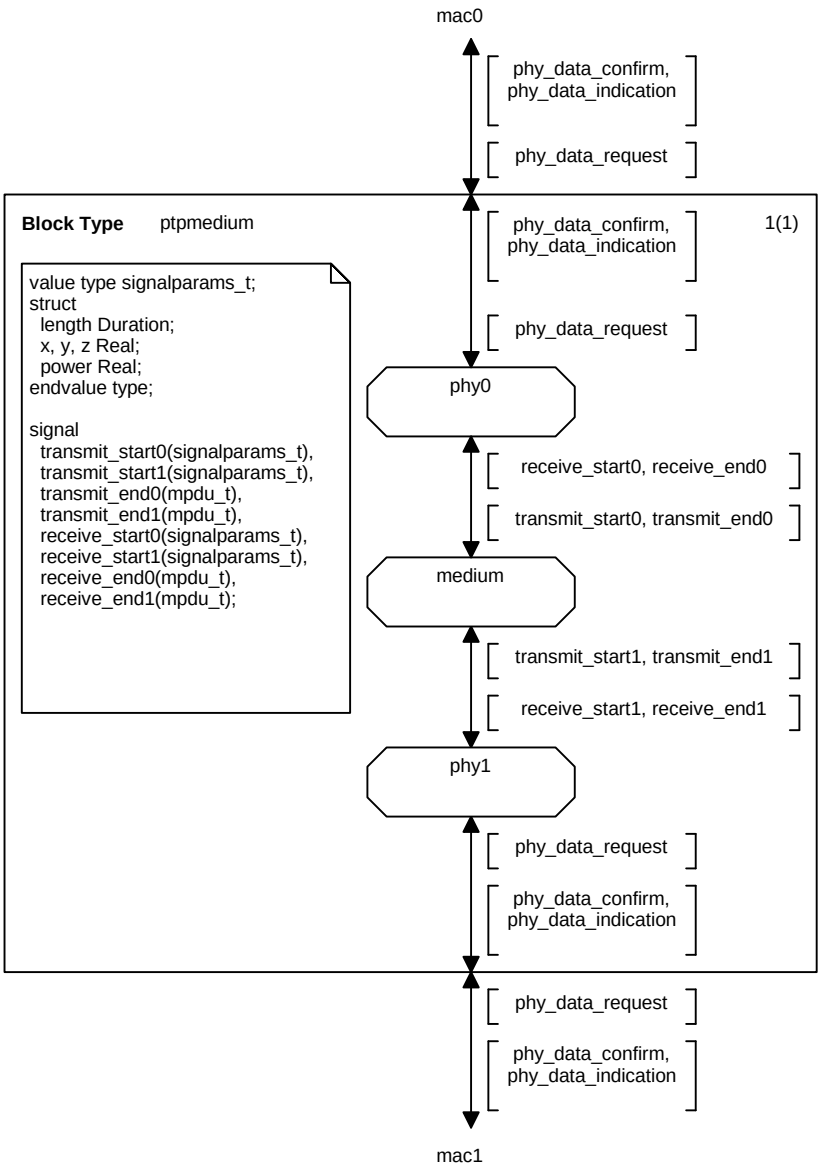


Figure 5.6: Model of a Point-to-point Wireless Medium

within Figure 5.6 as the `medium` block. When a `transmit_start` signal is issued to this block, it is relayed to the opposing physical layer model in the form of a `receive_start` signal. Correspondingly, `transmit_end` is converted to `receive_end` except where overlap of transmissions is detected in which case the `receive_end` signal is withheld.

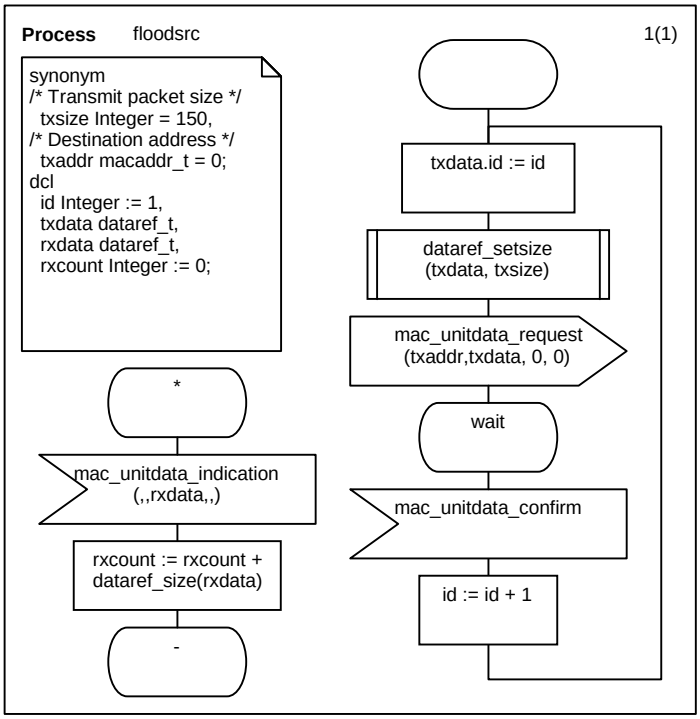


Figure 5.7: Flood Traffic Source

5.1.2 Traffic Models

Traffic models are used at the SDL simulation stage to provide stimulus for the protocols under development, allowing analysis of the frame exchange sequences and internal protocol communications. Given the goals of the SDL simulation stage set out earlier, a high level of detail is not necessary in these traffic models. In particular, for the majority of purposes it is unnecessary to include detailed information about MSDU payload. To this end, the SDL implementation of the data reference abstraction is a structure type. Arbitrary data can be included in this structure, but as a base it provides for payload length and unique identifier elements, each of type `Integer`. SDL operators and procedures implement the data reference operations.

Two basic traffic models for use in SDL simulation of MAC protocol specifications are presented here. The constant bit-rate traffic source provides fixed-size packets at a defined interval. The flood traffic source will provide fixed-size packets as quickly as the MAC protocol can accept them in order to

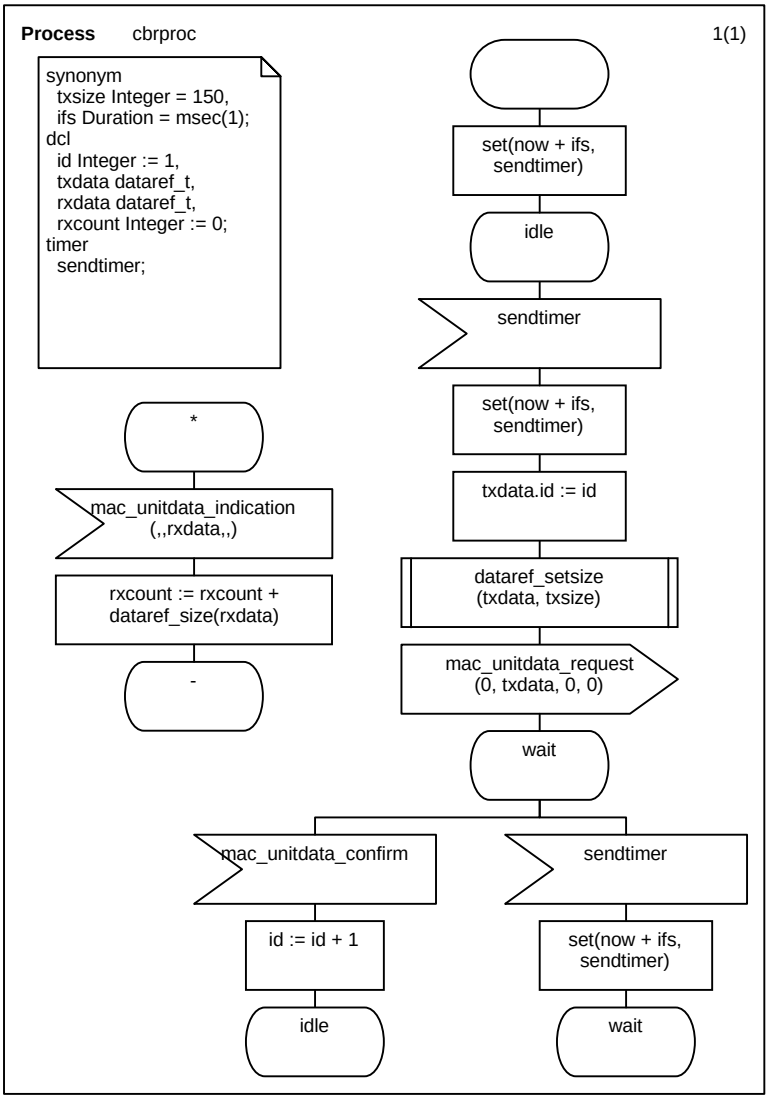


Figure 5.8: Constant Bit-Rate Traffic Source

test the protocol under load. The implementation of these models is as SDL agents with a single gate which is connected in simulation to the MAC service access point of the corresponding protocol node. As would be expected, these models deal in the `mac_unitdata_request`, `mac_unitdata_confirm`, and `mac_unitdata_indication` primitives.

Shown in Figure 5.7 is the flood source which will saturate its MAC with packets of a specified size. The source will also act as a sink for data, recording the total number of bytes received. A monotonically increasing payload identi-

fier is maintained which allows identification of packets in the frame exchange MSC. This model allows for automated basic stimulus of a protocol to test either single or bidirectional data transfer frame exchanges.

Figure 5.8 illustrates the constant bit-rate traffic source. This model is similar to the flood source, but the rate of issue of `mac_unitdata_request` primitives is limited by the fixed period of the send timer.

5.1.3 Performing Simulations

The Cinderella SDL environment allows for simulation of an SDL *system* – the top-level agent. For the protocol simulations that we seek to perform, the system consists of: multiple instances of our protocol under test; a physical layer model to allow communication between the nodes; and traffic sources and sinks to provide stimulus for the simulation. Figure 5.9 shows an initial testbench for the PTPMAC protocol. This testbench incorporates a constant bit-rate traffic source and a reliable point-to-point medium to allow basic testing of the protocol.

Integration of developed MAC protocols into this testbench is a straightforward process, with the developer able to graphically configure the network topology within the Cinderella environment. Appropriate traffic and error models can be applied as discussed in Sections 5.1.1 and 5.1.2.

The simulation result can be viewed as a MSC to allow rapid visual analysis of protocol operation. An excerpt from the MSC generated through SDL simulation of the PTPMAC protocol is shown in Figure 5.10. Visible within this MSC are five axes corresponding to the flood traffic source (`src1`); the MAC which it is attached to (`mac1`); the second MAC which is the traffic target (`mac0`); the block encapsulating the physical layer and medium models (`air`); and the environment (`env`).

Exchanges between the `src1` and `mac1` blocks show the interactions through the MAC service access point required to initiate the transfer of MSDUs. The

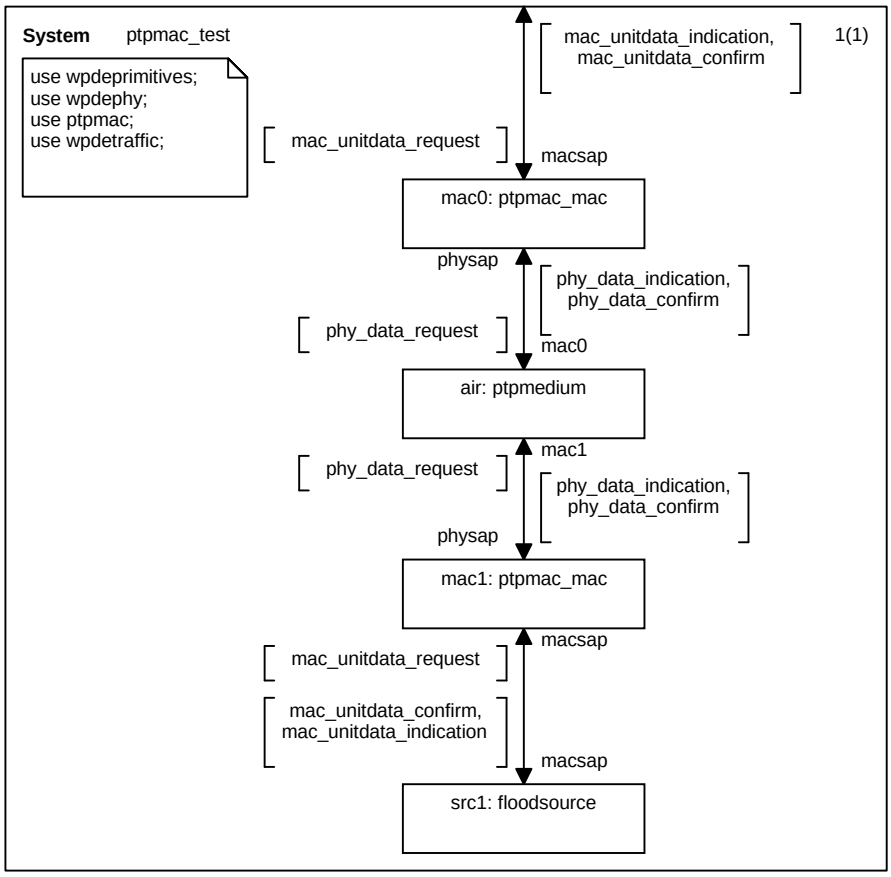


Figure 5.9: SDL Protocol Testbench

first activity on the `mac0` axis shows that it wins the transmit timeout race which is in effect from the start of simulation. Thus the first occurrence of a `phy_data_indication` primitive is delivered to the `mac1` instance and can be seen to be a pure token with neither acknowledgement nor data.

The MPDU returned by the `mac1` instance at this point carries the first MSDU generated by the traffic source `src1`, and can be seen to be successfully delivered from the `mac0` instance to the environment (`env`). The corresponding `mac_unitdata_confirm` is not returned to `src1`, however, until acknowledgement is received at `mac1`. It can be seen here that the subsequent `mac_unitdata_request` issued from `src1` to `mac1` is received after the token transmission has been initiated. This exposes an inefficiency in the current PTPMAC protocol operation: the time between transmission of two consecutive MSDUs from a particular MAC entity is always at least the duration

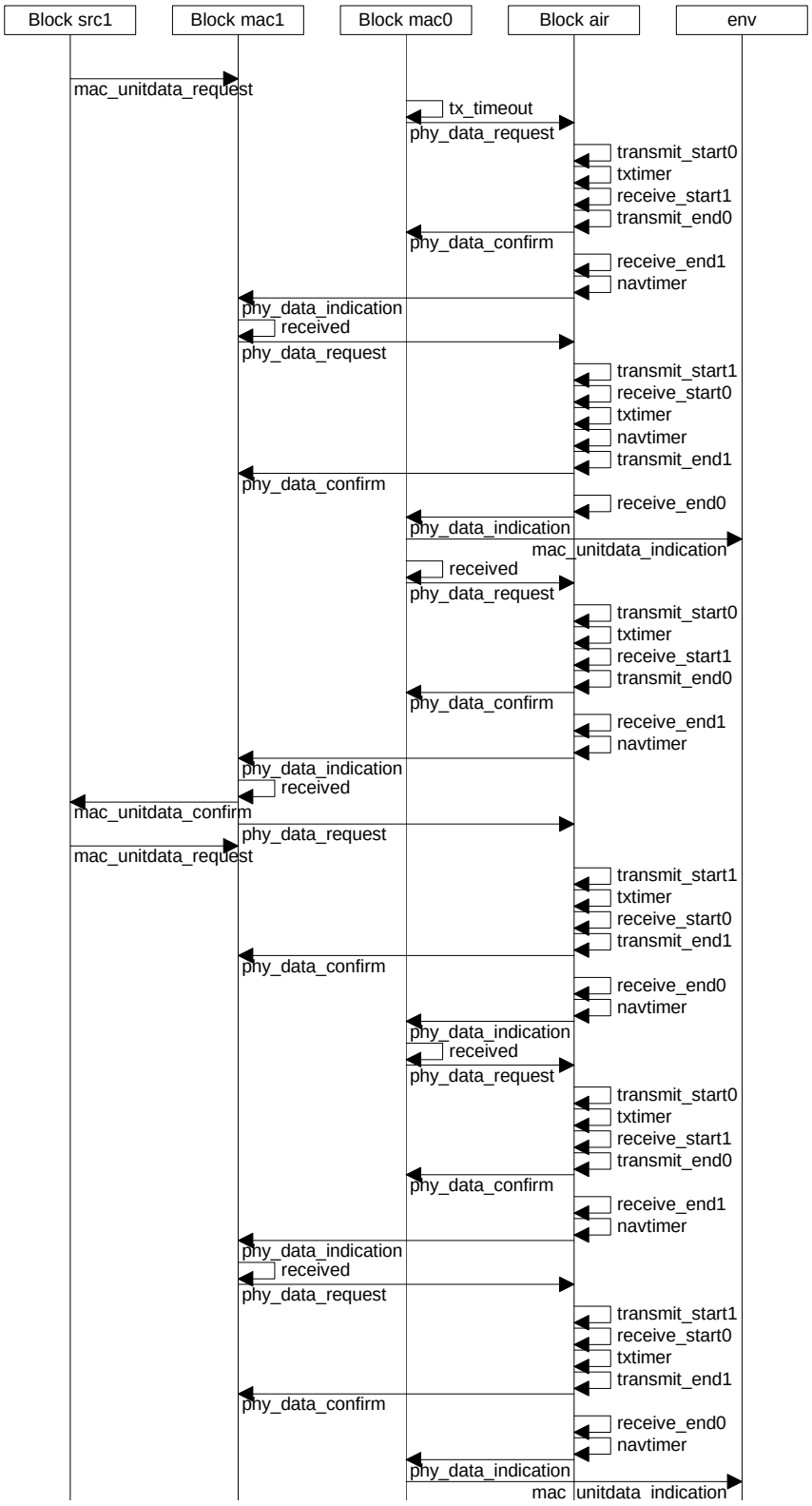


Figure 5.10: Initial Simulation of the PTPMAC Protocol

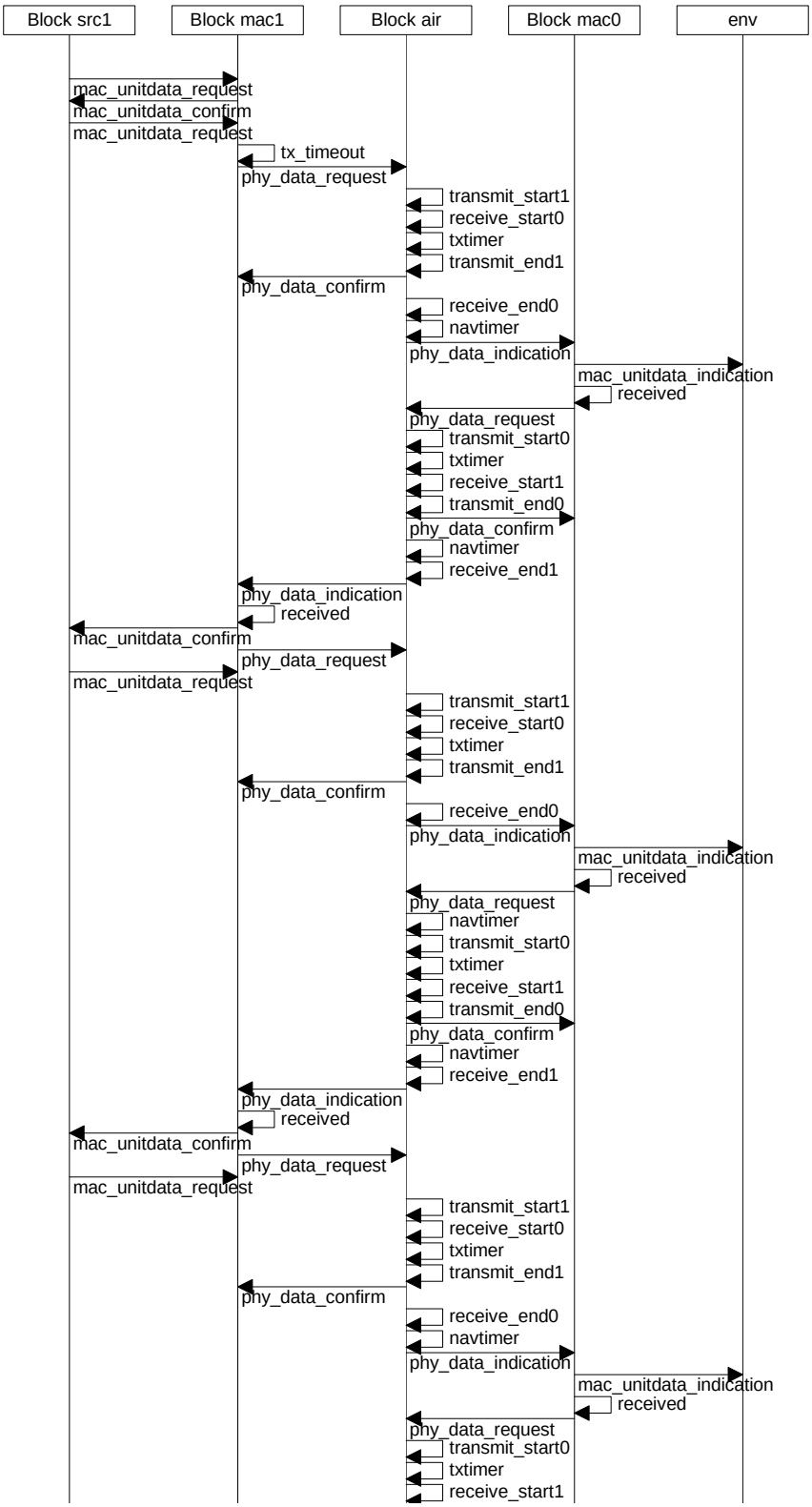


Figure 5.11: PTPMAC MSC with Double-Buffering

required to exchange the token twice.

This behaviour is due to the strict flow control enforced by the issue of the `mac_unitdata_confirm` primitive on receipt of the data acknowledgement. As receipt of the packet containing the acknowledgement will cause a transmission to occur immediately, the traffic source will not have had a chance to provide another MSDU for transmission. This illustrates the need for some degree of buffering of transmit MSDUs.

The amount of buffering that is required for optimal operation in a real system will be dependent on a number of factors including the characteristics of the host interface, driver implementation and host processing power. As a first attempt at improving the efficiency of the protocol, we extend our transmit unit to support buffering of two packets. Figure 5.11 shows the MSC generated through simulation of the double-buffered model of the PTPMAC protocol. While holding an initial packet in case retransmission is required, the MAC may now buffer the next packet to be sent so that when positive acknowledgement is received the following transmission can carry the next data packet. This dramatically improves the efficiency of the protocol.

5.2 Network Simulation

Though SDL simulation allows the verification of frame exchange sequences, the lack of model detail makes it unlikely that simulation results will give any accurate information on protocol performance in the real world. One way to allow this would be to improve the accuracy of the SDL models, however this is not trivial, as the acceptable abstractions are generally unclear [18]. Modern network simulators incorporate much of the knowledge in this area, and therefore typically provide the most representative network elements short of an actual implementation. Such simulators are generally highly optimised for performance, and will allow rapid exploration of a wider range of model and environment parameters than would be possible within a general purpose SDL simulation environment. The use of network simulation at this stage of

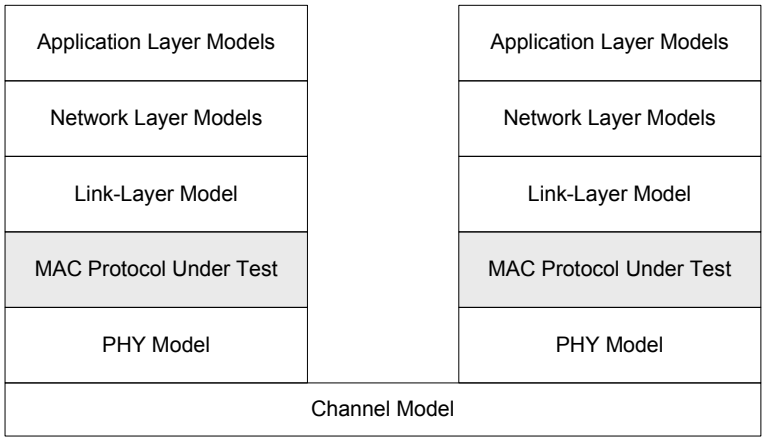


Figure 5.12: Network Simulation Elements

the design process can enable problems to be identified and corrected early to avoid the increased cost of addressing them at later stages of the design cycle.

As with SDL simulation we must consider the network simulation testbench, an example of which is illustrated in Figure 5.12. The added elements within the network simulation framework include more detailed models at several higher layers of the network stack, along with more representative physical layer and channel models. Several network simulators are available that would be suitable for use in this work. The Network Simulator – *ns-2* – is a high-performance discrete event simulator targeted at network simulation. It consists of a compiled C++ core, coupled with an OTcl interpreter front-end. The scheduler and other processor intensive aspects of the simulation are implemented in C++ for efficiency. This includes core functionality of the simulation models. The OTcl front-end allows flexible configuration of simulations through scripts interpreted at runtime.

Although *ns-2* was originally designed for use in wired networks, its underlying structure is sufficiently flexible to make it a powerful tool for many aspects of network simulation, including the analysis of wireless MAC protocols. Research in the CMU Monarch group [85] has produced extensions to *ns-2* for supporting simulation of mobile networking scenarios. This work includes a simulation model of the IEEE 802.11 protocol, and radio propagation models based on the Friss free-space and two-ray ground reflection models. A third

makes use of the more realistic shadowing model which takes into account the effects of multi-path propagation.

The *ns-2* simulator is chosen as the basis for this aspect of the proposed development framework due to its good range of protocol and wireless channel simulation models, flexible configuration front-end, and widespread use within the network research community [23]. Early integration to *ns-2* gives access to other powerful extensions and tools based around that platform such as iNSpect [94], the Network Simulation Cradle [95] or a range of others [96].

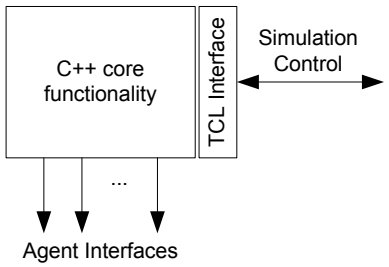


Figure 5.13: Interfaces of an *ns-2* Object

5.2.1 The *ns-2* Simulation Model

Simulations in *ns-2* are built from a structure of interacting objects forming a representative model of the system under test. The typical simulation object in *ns-2* is constructed using a C++ class, with an OTcl interface allowing runtime configuration. Figure 5.13 depicts this structure. The core model within each object may have any number of interfaces which allow communication with other objects. The OTcl interface allows dynamic instantiation of objects and the connection of their interfaces. In this way, a simulation consisting of a number of communicating objects can be created using a simple script.

Granularity of *ns-2* simulation objects is such that generally several are required to provide the complete functionality of a network node. Figure 5.14 illustrates the structure of the *ns-2* mobilenode, which brings together the simulation objects necessary to model accurately the lower levels of the network stack and their interactions. The Link Layer provides modelling of the Address Resolution Protocol (ARP) in cooperation with the ARP Table. Once

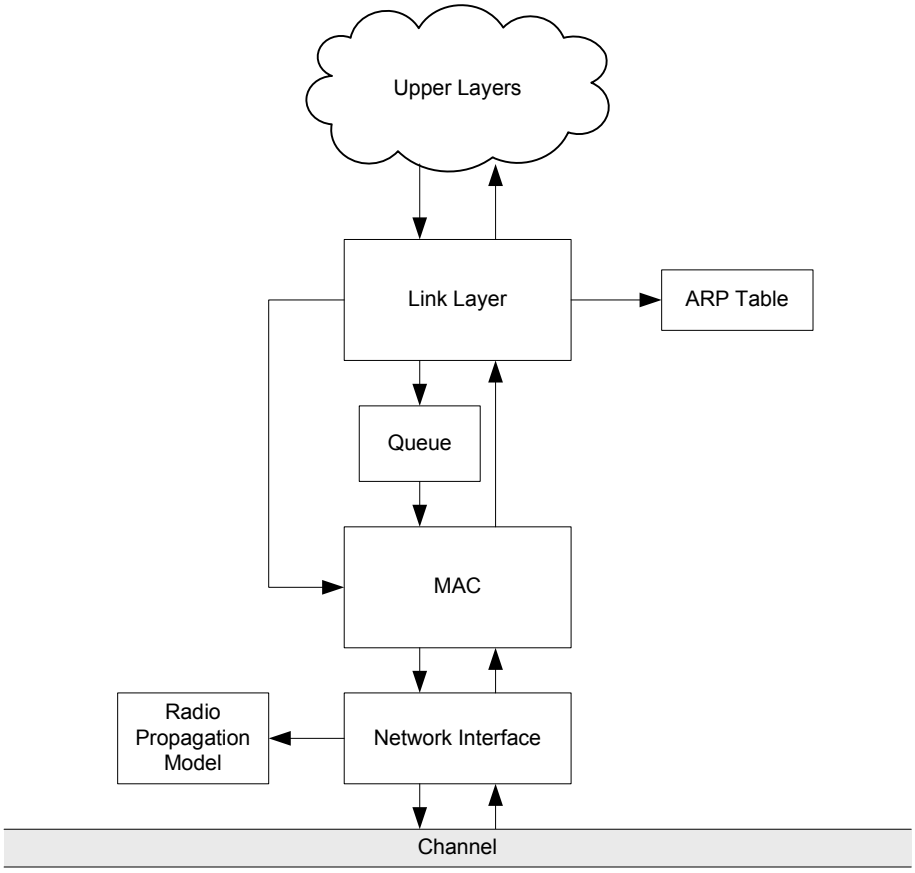


Figure 5.14: Architecture of the *ns-2* Mobilenode

a destination MAC address has been determined, outgoing data-link packets are queued to the MAC. The Queue may operate using a range of algorithms, from a simple First-in, First-out (FIFO) with drop-tail, through to a more sophisticated class-based algorithm which provides for simulation of differential services. The MAC is clearly the sub-layer that we are concerned with and this will be discussed in more detail later. The Network Interface object models the physical layer, and applies the Radio Propagation Model. The outgoing packet is passed to the Network Interface by the MAC, and then is distributed via the Channel. On the receive path, the procedure is reversed, with the subtle difference being that packets are passed directly from MAC to Link Layer rather than through a queue object.

The Radio Propagation and Channel objects collectively model propagation delay between nodes and signal attenuation to the receiver using either the free-

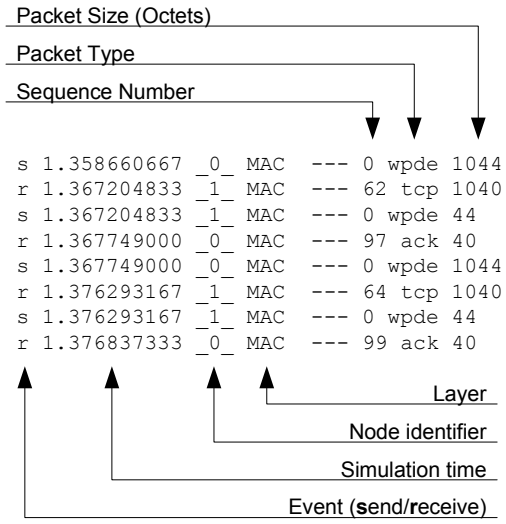


Figure 5.15: Extract of *ns-2* Simulation Trace

space, two-ray ground, or shadowing models, with bit errors, interference and capture modelled at the receiver in the Network Interface. This architecture clearly separates the MAC from this physical layer modelling, allowing new or modified protocols to readily make use of these facilities.

Transport layer protocols are modelled within *ns-2* as a special class of object known as a transport agent. Numerous agents are provided with the standard distribution including models of several variants of TCP,UDP, and Real-time Transport Protocol (RTP). Like other simulation objects, transport agents may be configured at run-time using OTcl. Application objects sit above transport agents in the simulated stack, and provide modelling of traffic characteristics. Supplied application objects provide modelling of FTP, Constant Bit Rate (CBR) traffic such as VoIP, or playback from a captured or synthesised packet trace file.

Two general methods of drawing results from the simulation are possible: simulation tracing and collection of statistics within models. Trace simulation objects are provided which act as a packet sink, and record the timing and details of these packets to a file. Post-processing of the recorded trace allows analysis of protocol features. Figure 5.15 shows an excerpt from the MAC layer simulation trace of the PTPMAC protocol described within the WPDE framework. Each line within this trace represents an event associated with a

MAC object within the simulation. This trace shows a TCP session between two nodes. The true nature of the packets (TCP data and acknowledgements) is only visible in the trace entries associated with the receiving node, due to the location of the trace points at the network interface in the downward direction (send) and at the interface to the link-layer in the upward direction. Sent packets are traced after encapsulation into their MPDU format, and therefore their original type is hidden. Comparing the packet size in octets of each send/receive pair of trace entries it can be seen that the MPDU (indicated by a packet type of *wpde*) is four octets longer than the corresponding TCP packet – this is the PTPMAC protocol framing overhead.

The alternate means of obtaining results is through collection of statistics within simulation models. These statistics can be extracted through the OTcl interface at arbitrary points within the simulation. This approach is commonly used in simulations involving TCP or UDP agents, with the agent maintaining a variable which records the count of bytes received, and a simulation script sampling and then zeroing this variable at fixed intervals through the OTcl interface, in order to track throughput across simulation.

5.2.2 Integrating WPDE MAC Protocols

Ns-2 provides a powerful environment for performing network simulations with a rich library of simulation models available to the network researcher. Though *ns-2* provides a fairly moderate learning curve for the development of simulation models, manual implementation of simulation models for a protocol under development is not desirable. As well as being time-consuming, manual implementation or translation also presents the chance of human error corrupting the model and invalidating any results that simulation may produce. Consistency of representation between different stages of the design process [17] is a key goal of the work described in this thesis, and although different representation techniques may be required at different stages, automated translation can remove human error from the process. The tool presented in Chapter 4 for translation of SDL protocol models into C++ makes this approach feasible given a suitable surrounding framework. In order to provide a framework

for integration of WPDE MAC protocols into the *ns-2* simulator, it is first necessary to consider how we may map the channels defined in Chapter 4.

Channels

The channels defined in Chapter 4 provide the standard interfaces for interactions between MAC protocols developed within the WPDE framework and their environments. In integrating protocols with the *ns-2* simulator, these channels must be mapped to those available to native *ns-2* MAC models.

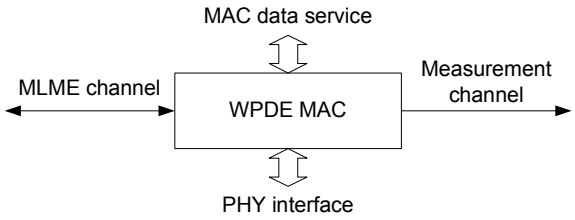


Figure 5.16: Interfaces to the WPDE MAC

Figure 5.16 illustrates the four defined channels. The data service provided by the MAC clearly must interact with the link-layer and queue elements in the *ns-2* mobilenode. The issue of a packet from a higher layer results in injection of a `mac_unitdata_request` signal to the MAC protocol. Similarly, data indications generated by the MAC cause a packet to be conveyed to the link-layer.

Like the MAC data service channel, the physical layer interface defined in Chapter 4 is on the core data-path through the MAC protocol. Within the *ns-2* model, the physical layer interface ultimately interacts with the network interface depicted in Figure 5.14, though it is necessary to appropriately model the transceiver at this interface.

The MLME channel provides for manipulation of MAC parameters and control of MAC operation. These functions are typically invoked relatively infrequently in comparison to the MAC data service primitives, and are not directly controlled by the upper layers of the network stack. For this reason these are mapped to the OTcl interface within the *ns-2* framework. This approach en-

hances the control available to the user, allowing scripting to precisely control MLME access and thus the protocol under test.

The measurement channel is intended to allow precise correlation of internal MAC events with those that are more readily observed externally. Of particular interest is the relationship of emitted measurement signals to interactions on the MAC data service and PHY interface channels; however, measurement signals should not enter any standard data paths. The standard *ns-2* MAC trace file provides a convenient destination for these measurement signals, where their relationship to other trace events is apparent. Signals emitted on the measurement channel generate a trace file entry which includes the current simulation time, along with the measurement point identifier and argument.

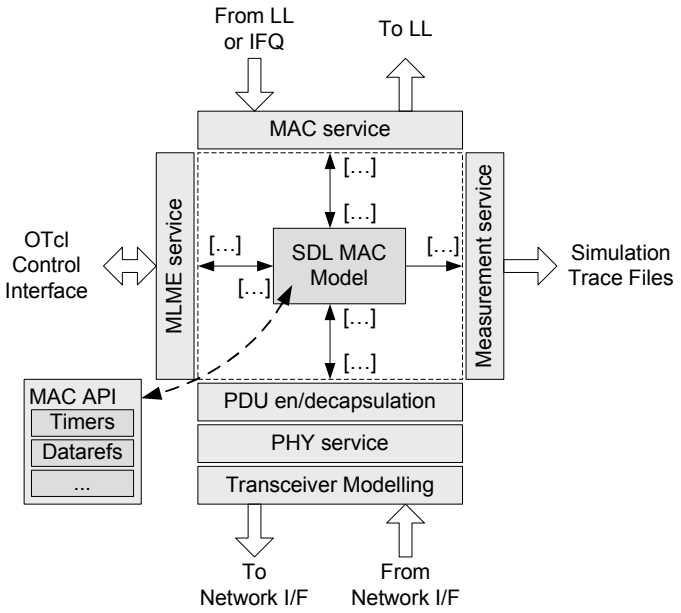


Figure 5.17: Architecture of the *ns-2* WPDE Environment

Figure 5.17 illustrates the integration described above of an SDL protocol model into the *ns-2* environment. The MAC, MLME, Measurement, and PHY service blocks implement the mappings described above. The MAC API is used by the protocol model to provide local implementations for features such as the timer and data reference mechanisms. Two other blocks are apparent between the physical layer interface of the MAC and the *ns-2* network interface: the MPDU en/decapsulation, and transceiver modelling blocks. These are discussed in the following sections.

PDU Encapsulation

The physical layer interface primitives defined in Chapter 4 are designed to convey MPDUs to and from the PHY. The parameters of these primitives are specific to the protocol under development, reflecting the various fields of the specific MPDU format. At early stages of design, no detail is required or assumed about the actual layout or overhead of MPDUs when serialised by the transceiver, however at the stage of network simulation this detail may begin to become relevant. The framework depicted in Figure 5.17 supports the specification of MPDU encapsulation and decapsulation routines by providing shell C++ functions within which the data reference API routines can be used to pack and un-pack MPDUs from and to their constituent individual fields.

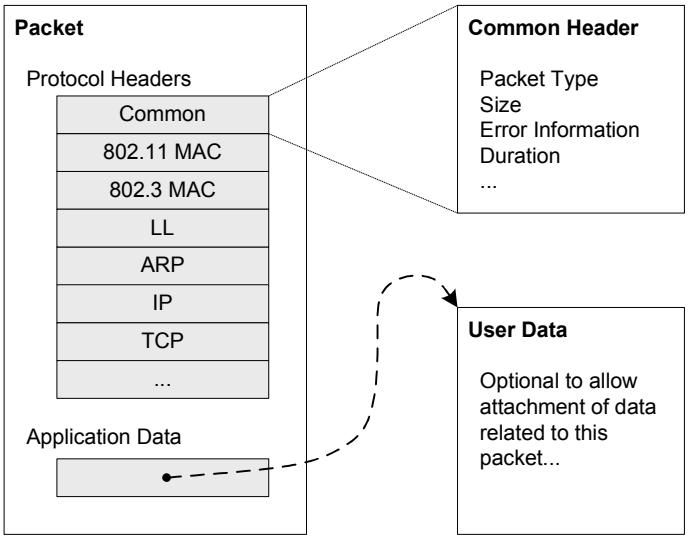


Figure 5.18: The Structure of an *ns-2* Packet

The *Packet* (illustrated in Figure 5.18) is the basic unit of data exchange between protocol elements in an *ns-2* simulation, and so it is into this form that MAC protocol transmissions must be converted. The primary components of interest in a *ns-2* packet are the protocol headers and the optional application data reference. Each packet has allocated space for protocol headers to cater for all possible protocols within the system. A common header field provides basic information for the packet including the protocol it contains, payload size, duration on the air, and also holds flags for marking packets with error. Payload data is generally ignored within an *ns-2* simulation in order to

maximise speed, however scenarios such as emulation using live data require that this be tracked and so the application data field of each packet provides a means for attaching arbitrary data to a packet.

Two approaches are possible to the encapsulation of custom protocol information within the *ns-2* packet. The first requires the definition of a custom protocol header containing variables which correspond to the specific fields of the protocol MPDU. Encapsulation would then consist of the direct copy of `phy_data_request` signal parameters into the corresponding header fields. Alternatively, an accurate MPDU could be constructed and attached to the packet using the application data field. This second method has the advantage that it can be carried over verbatim to an implementation, and for this reason is the method chosen here.

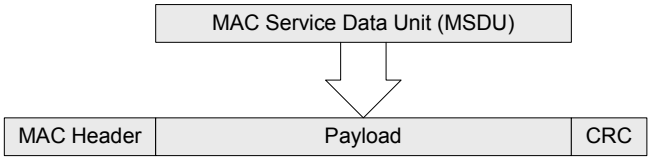


Figure 5.19: MSDU Encapsulation within an MPDU

A common task in the creation of an MPDU is the encapsulation of an MSDU originated from a higher layer. A typical example is illustrated in Figure 5.19 where the MAC has prefixed the MSDU with a header, and appended a CRC error check to create an MPDU. Within the simulation environment it is important to optimise this potentially intensive process in order to improve performance for large scale simulations. This can be achieved by the insertion of a reference to the packet being encapsulated as opposed to full encoding of the payload.

Figure 5.20 illustrates this approach, where a higher layer packet is encapsulated by reference between a header and CRC within an MPDU. This method provides an extremely efficient solution to encapsulation, as no payload copy is required. The approach is also suitable for use with arbitrary higher-layer protocols, as no knowledge of packet internals is required.

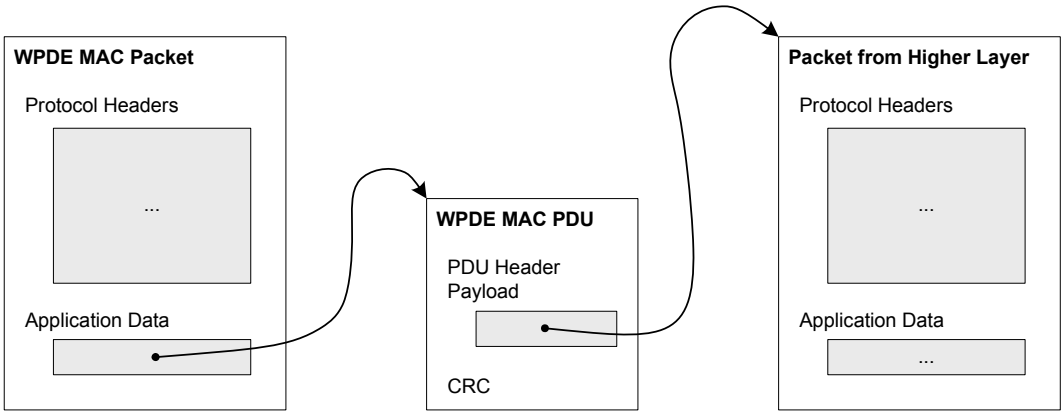


Figure 5.20: Encapsulating Packets within PDUs

Modelling the Transceiver

Modelling of the wireless transceiver in *ns-2* is the responsibility of the corresponding MAC simulation object, and hence must be provided for in the framework which will encapsulate our WPDE protocols. The transceiver model included with the *ns-2* 802.11 MAC implementation provides a suitable basis this.

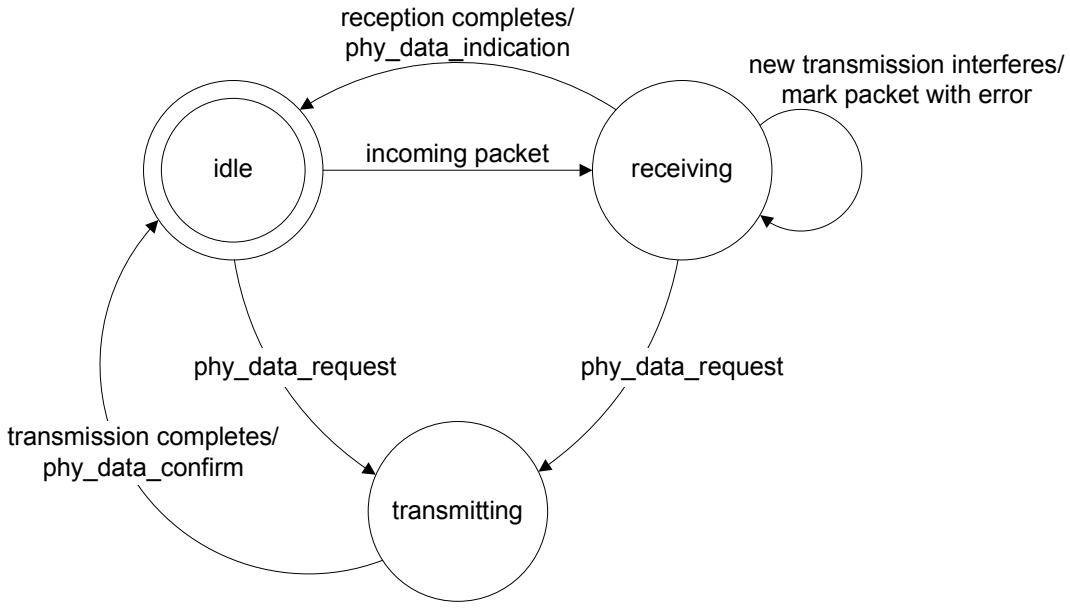


Figure 5.21: The *ns-2* Radio Model

Figure 5.21 illustrates the basic radio model used. The radio is considered

to be in one of three states of operation: `idle`, `transmit`, or `receive`. The transmission process is initiated by the emission of a `phy_data_request` signal by the MAC following the steps listed in the pseudo-code below:

1. The MAC emits a `phy_data_request` signal.
2. On-air duration is computed and inserted into the packet data structure.
3. Packet is passed via the network interface to the channel for distribution.
4. A timer is set for transmission completion.
5. On expiry of the timer a `phy_data_confirm` is passed to the MAC.

Modelling of the RF environment is performed by the network interface and channel objects. Propagation delay is modelled by the channel in its scheduling of the receive event at the remote nodes. In reception the chosen propagation model is used to apply errors representative of multi-path effects. The received packet is marked with its received signal strength indicator which may be used to model capture. The following sequence shows the MAC receive procedure at each node.

1. A packet is passed up from the network interface.
2. The channel state is marked as becoming busy.
3. If there is no existing activity then:
   (a) A timer is set for completion of the reception.
   (b) On timer expiry a `phy_data_indication` is passed to the MAC.
4. If there is existing activity then:
   (a) If the power level of the new packet is within the capture threshold, then the packet in reception is marked as having an error.
   (b) The new packet is dropped.

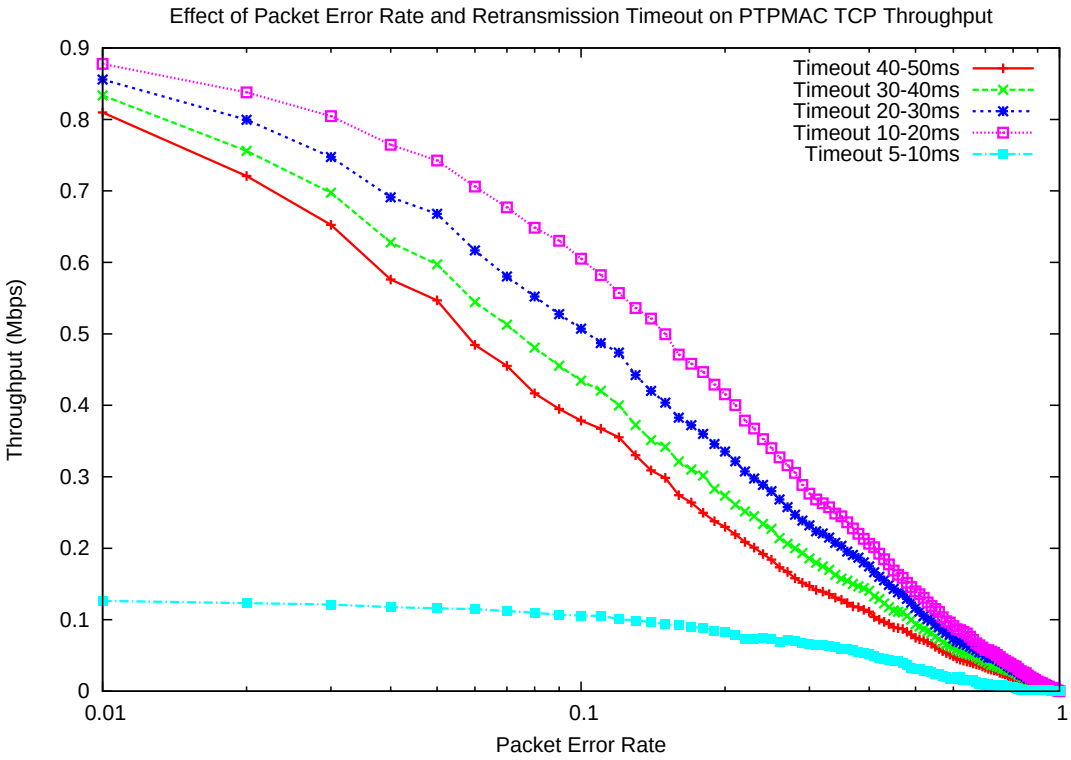


Figure 5.22: Effect of Packet Error Rate on PTPMAC TCP Session

This approach to the modelling of the physical layer provides an extremely efficient implementation that is suitable for most research. Two main limitations exist – no provision is made for re-capture of the receiver by a second signal, and no received signal strength information is available for signals that are not being tracked by the receiver. These limitations do not, however, invalidate this model for use in the majority of simulation scenarios.

Figure 5.22 shows results collected from *ns-2* simulation of the PTPMAC protocol, indicating the effect of the packet error rate on the throughput of a TCP session using several different retransmission timeout windows, and a maximum serialisation rate of 1Mbps. These results show a maximum TCP throughput of 917Kbps in the absence of packet errors. The results for a retransmission window of 5-10ms shows throughput significantly lower than the others. This is indicative of the fact that many timeout values within this window will be less than the serialisation time of a typical TCP segment at 1Mbps. This shows the need for appropriate selection of the timeout window.

Though not a robust experiment, this example shows the ease with which developed protocols can be integrated to simulation using the described framework. The simulation was conducted using the WPDE model automatically generated from the SDL description of PTPMAC, with the topology consisting of two stationary co-planar PTPMAC nodes separated by 50 metres using omni-directional antenna and the *ns-2* two-ray ground propagation model. Results from one run per packet error rate were recorded to form the graph in Figure 5.22.

Chapter 6

Implementation Platform

Chapter 5 discussed the need for a coherent approach to the evaluation of protocols and presented simulation testbenches which have been developed to provide debugging and performance analysis of protocols designed within the WPDE. Simulation provides a powerful tool for evaluation as long as care is taken to choose suitable methods of modelling higher layers of the network stack [18]. Similarly realistic physical layer models are required to ensure the validity of simulation results [16, 51, 97]. Prototype implementation of protocols allows evaluation with actual traffic and a real physical layer. This provides a greater degree of confidence in the evaluation results, and hence allows validation of both the protocol and simulation models. It is desirable to provide for prototype implementation and evaluation of protocols developed within the WPDE.

Several key characteristics are required of an implementation platform: a fundamental requirement is that the platform support implementation of realistic modern protocols such as those discussed in Chapter 2. This requires appropriate physical layer and host interfaces, and sufficient capacity for configuration to ensure that MAC functionality can be captured in its entirety; though a majority of MAC functionality can be provided by embedded software, to support aspects with tight timing constraints the platform should also support mixed hardware and software implementations; Chapter 5 identified the need to eval-

uate protocols with representative traffic. For the implementation platform this means deployment of the protocol-under-test into real-world application scenarios. Wireless technologies are often deployed in mobile or remote locations so the implementation platform form-factor must allow portability to ensure its usefulness in such situations; in keeping with the outright aim of the work described in this thesis, it is desirable to minimise the effort required to produce a prototype implementation. By keeping this process short, implementation and measurement may be used at an earlier stage of the design lifetime. The provision of library components for commonly used blocks of hardware is one means of addressing this goal; finally, the implementation platform must provide for the measurement and evaluation of protocols which have been implemented. This is primarily discussed in Chapter 7, but must also be considered in the design of the implementation platform.

Few wireless protocol implementation platforms are readily available to the academic community. Ganz et al. [38] describe a system using a commercial wireless NIC running custom firmware and a Windows device driver which allows the incorporation of MAC functionality on the host. This approach provides a good degree of flexibility in design, but restricts performance due to inter-packet turn-around being constrained by host interface and driver scheduling latencies. No hardware protocol assistance is possible within this approach, and support for measurement is not apparent.

Recent work in the open-source community has produced the Multi-band Atheros Driver for Wi-Fi (MADWIFI) [41], which runs on various Linux operating systems and supports the majority of cards based on Atheros chipsets. The driver incorporates a binary (closed source Atheros proprietary) Hardware Abstraction Layer (HAL), which provides an API allowing access to relatively low level chipset functionality. Several projects [42, 43, 44, 45] have implemented custom protocols using the MADWIFI driver. MADWIFI improves on the system described in [38] by making use of optimised hardware within the chipset, and a more flexible API; however, packets are still restricted to using some 802.11 framing at the MAC level, and host interface latencies still contribute a significant overhead – particularly at the higher bit-rates which are supported.

The TUTWLAN demonstrator [50] incorporates many of the features which are desirable in an implementation platform. It uses a combination of FPGA and DSP to provide MAC implementation, allowing for low-latency timing constraints to be met. A commercial 802.11 MAC-less reference design is used as the separate radio module. Several improvements must be made to the basic TUTWLAN architecture to provide an implementation platform which meets the goals set out above. The form-factor of the TUTWLAN platform with a full Peripheral Component Interconnect (PCI) interface and multiple circuit boards precludes its deployment in mobile or low-power nodes. Integration of the functionality onto a circuit board suitable for use with existing low-power single-board computers such as those produced by Soekris Engineering [98], will allow flexibility in choosing an implementation scenario for prototype protocols. The TUTWLAN system provides for mixed hardware and software design through the interface between the FPGA and DSP chips. Though this allows for information exchange between the elements of the partitioned design, the interface in this case is inflexible. Modern FPGA technology provides for on-chip integration of microprocessors blocks, allowing extremely close and flexible coupling of the hardware and software aspects of a MAC protocol. Finally, the described TUTWLAN platform does not provide integrated support for debugging and measurement of protocols in operation. As discussed later in Chapter 7, this is a key part of the development process.

To address these requirements, the hardware, firmware, and software for a wireless MAC protocol implementation platform has been developed as part of the work described in this thesis. The remainder of this chapter details the key elements of this platform and its role in supporting the design process.

6.1 The WAG device

The Wireless Analysis and Generation (WAG) device (shown in Figure 6.1) was designed as part of the work described in this thesis to allow the implementation and evaluation of wireless protocols. It combines an 802.11b [2, 56] radio chipset and an FPGA with integrated processor onto a Mini-PCI inter-

Figure 6.1: WAG v2.1 Circuit Board

face card. A block diagram depicting the hardware structure can be seen in Figure 6.2.

FPGA technology provides an ideal basis for a wireless protocol implementation platform. By designing using hardware description languages such as the VHSIC[1] Hardware Description Language (VHDL) or Verilog, the design cycle can be minimised, allowing efficient implementation of timing critical MAC functionality. In contrast, some MAC operations are best described in an imperative form and implemented as software running on an embedded processor. The chosen FPGA provides for this through an integrated PowerPC [99] block which can be tightly integrated with the other hardware. On-board LEDs provide a visual indicator of MAC activity, and a processor debug interface provides a powerful development tool.

The form-factor of the card allows for designs under test to be deployed in mobile and remote environments. Single-board computers such as the Soekris [98] running the Linux operating system can be used with the WAG to provide a low-power portable wireless development system.

[1] VHSIC: Very High Speed Integrated Circuit

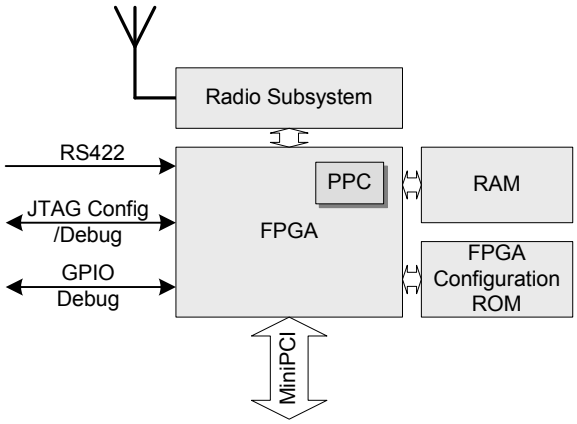


Figure 6.2: Hardware Structure

6.1.1 Radio

The selection of physical layer technology was an important decision in the design of the WAG devices. Three key factors were involved in the choice of radio chipset: the described implementation platform would support any of a range of possible radio technologies. It was important, however, to use a technology with a suitable degree of performance to allow implementation of representative modern protocols; in order to support MAC layer design it was necessary to use a chipset which provided a device boundary between the MAC and modem functionality. Trends toward increasing levels of chipset integration meant that this restricted the range of choices available; the final requirement in selection of radio chipset was that it be readily available in the small quantities used within this work. For these reasons the RF Micro Devices [100] chipset consisting of an 802.11b DSSS baseband modem, transceiver, power amplifier, and RF switch was chosen for the WAG radio. Figure 6.3 provides a simplified illustration of the WAG radio architecture.

The RF5189 power amplifier, RF2436 RF transmit/receive switch, and a 2400-2500MHz bandpass Surface Acoustic Wave (SAW) filter make up the front end of the radio architecture. The RF2958 transceiver incorporates both RF and IF (intermediate frequency of 374MHz) local oscillators, and provides Quadrature Phase Shift Keying (QPSK) modulation and demodulation between baseband and IF, up and down conversion between IF and RF signals, low-noise am-

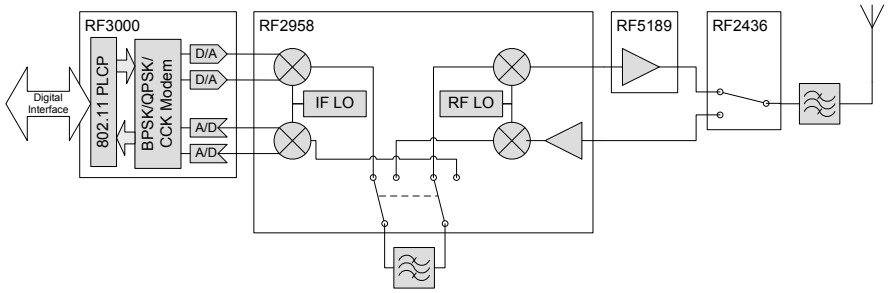


Figure 6.3: WAG Radio Architecture

plification of the received signal, and both receive and transmit gain control. The external IF filter is shared between the transmit and receive paths. This filter provides the channel selectivity in the receive path, and in the transmit direction acts to reduce spurious emissions in adjacent channels.

The RF3000 baseband modem provides the spread-spectrum modulation of a digital bit-stream, and the respective demodulation. It includes an equaliser to ease the effects of multi-path propagation which can impact on the performance at higher data rates. Physical layer framing is applied as per the IEEE 802.11b PLCP. Within the modem are digital-to-analogue converters to provide transmit and receive variable gain control, with the latter controlled automatically to maintain appropriate level at the receive analogue-to-digital converters. The modem provides a CCA output which may be based on detected channel power, receiver acquisition, or both. The nature of the interface to the RF3000 device is crucial to the WAG design as it must provide for the physical layer service primitives set out in Chapter 4.

6.1.2 Digital Framework

The digital sections of the WAG architecture encompass the configurable logic and embedded processor which allow MAC implementation, along with fixed and configurable hardware providing for timing, debugging, and general support around the MAC subsystem. The primary component providing this functionality is the FPGA.

The FPGA is of the Xilinx Virtex-II Pro [101] family, providing for high-speed

firmware designs, which may incorporate the on-chip IBM PowerPC processor block [99]. A firmware framework has been implemented which supports MAC developers by providing an abstraction of the PCI and Radio interfaces that is more conducive to protocol development. The hardware description language VHDL has been used to implement this framework which is illustrated in Figure 6.4.

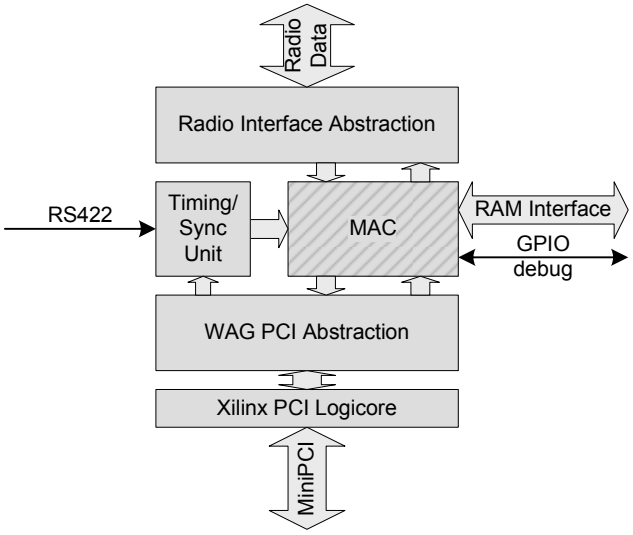


Figure 6.4: WAG Firmware Structure

The host bus interface is provided by purchased Xilinx intellectual property residing within the FPGA which deals with PCI bus transactions and provides a simplified interface for user applications. A transport mechanism has been implemented which provides FIFO streams of data towards the card from the host and vice-versa. To reduce the burden on host systems, bus mastering and burst transfer support have been implemented, maximising the efficiency of the data transfers. Circular buffers are maintained in the host memory, which the WAG transfers data to and from. An interrupt is signalled to the host on completion of a transfer.

On the radio side the data serialisation and de-serialisation is handled by the framework, which again presents two FIFO streams between the MAC and the transmit and receive units. Common radio control functions such as channel selection, transmit data-rate, and gain (serial interfaces at the radio chipset), are implemented by a radio control unit which abstracts the interface to a

parallel register write.

Non-volatile memory is necessary for operation of the embedded software which will implement aspects of MAC protocols. Though the FPGA provides some amount of internal Random Access Memory (RAM) which is available to the processor, this is likely to be insufficient for complex MAC designs and so further Static Random Access Memory (SRAM) has been provided externally.

Timing synchronisation is an important feature for supporting measurement of protocols. The RS422 serial interface allows for internal time synchronisation through an accurate Global Positioning System (GPS) pulse-per-second signal. This provides a mechanism for managing clock drift, and synchronising multiple nodes to allow distributed measurements. This is further discussed in Chapter 7.

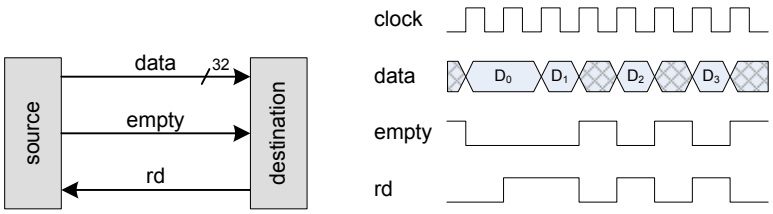


Figure 6.5: Internal Data Interface

The architecture shown in Figure 6.4 allows custom MAC implementations to be 'dropped in' by providing a clean and efficient interface which hides unnecessary host interface and radio detail from the system under development. Four primary data interfaces are visible in this block, providing data transfer between host and MAC, and MAC and radio. Each of these interfaces provides the unidirectional transfer of data using the signalling illustrated in Figure 6.5. Transfers are synchronised by the system clock, which operates at 50MHz. A valid transfer is indicated when the source de-asserts `empty` to indicate the presence of valid data, and the destination asserts `rd`. Data is latched into the destination entity at the completion of such a cycle. Asynchronous use of the `empty` signal feeding back through combinatorial logic into the `rd` input to the source is possible as the interface is only used within the FPGA where propagation delays are minimal. Though the maximum bandwidth of 1.6Gbps that this interface provides may seem excessive, this allows the flexibility of

optionally implementing arbitrary limits on bandwidth or latency to emulate an intended target host transport technology.

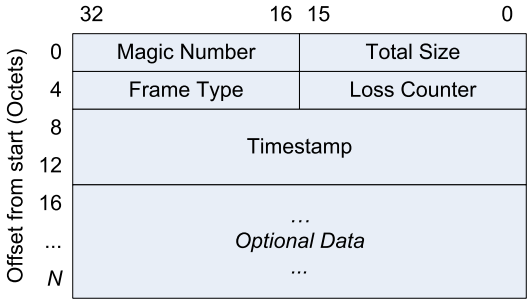


Figure 6.6: WAG Frame Format

Discrete units of data are transferred across these interfaces. These may be packets to or from the radio, or service primitives issued over the host interface. A consistent framing for these data units is defined to allow manipulation of data streams. A conforming unit of transfer over this interface is referred to as a WAG frame – a 32-bit aligned unit of data with the header format as illustrated in Figure 6.6. The first word of each frame contains the magic number `0xDAA1`[1], along with the total frame length in octets. This allows a frame processing element with limited internal state to readily identify frame boundaries within a data stream. In case of error, resynchronisation can be achieved by searching the stream for the magic number. The second header word contains frame type and loss counter fields. The frame type field implicitly specifies the format of the remaining frame body. Simple message frames may consist of solely the two-word frame header. The loss counter allows detection of frame lost due to insufficient bandwidth. Under normal circumstances this field will be zero, with a non-zero value indicating the count of frames lost since the previous frame in the stream.

The interface stages between MAC and host provide frame transport independent of the frame type. Any frame which has an appropriate first header word will be transferred to the host and made available to the driver. The radio however deals with specific frame types in order to encapsulate physical layer parameters and PDU data.

[1]This number corresponds to the author's University computer system user-name.

```
typedef struct frame_phy_data_indication_t {
  frame_t     hdr;         // common frame header
  timestamp_t ts;          // reception timestamp
  uint8_t     rssi;        // receive signal strength
  uint8_t     rxstatus;    // rx status bits from the modem
  uint16_t    length;      // payload length in bytes
  uint8_t     signal;      // 802.11PLCP signal field
  uint8_t     service;     // 802.11PLCP service field
  uint16_t    plcp_length; // 802.11PLCP length field (uS)
  uint8_t     data[0];     // placeholder for payload access
} frame_phy_data_indication_t;
```

Figure 6.7: PHY-DATA.indication Frame

Figure 6.7 shows a C structure detailing the format of the frames generated by the radio subsystem on reception of a packet. This frame can be considered equivalent to the PHY-DATA.indication signal defined in our generic MAC interface in Chapter 3. As always, the generic frame header as shown in Figure 6.6 lies at the start of the frame. A timestamp is taken corresponding to reception of the first symbol of the packet body, and this is made available within the frame. This timestamp is generated from the system timestamp counter. The following fields provide various reception parameters, including receive signal strength, modulation type and rate, and payload length. The start of packet data is aligned to a 32-bit boundary, and is padded to ensure the entire frame size aligns to 32 bits.

The frame sent to the radio subsystem for packet transmit is organised in a similar way. In particular the frame body length and the payload data are in the same position relative to the start of the frame. This allows the use of common or similar hardware units for data frame processing operations such as encryption or error control.

The control interface between the MAC and radio consists of three main signals. The `radio_pe` signal enables power to the radio circuitry, `radio_channel` allows specification of the 802.11-style channel of operation and `radio_txstart` is asserted to initiate transmission of a queued PDU. Similarly, four status inputs are provided from the radio subsystem to the MAC: `radio_receiving`

and `radio_transmitting` indicate the conditions of actively receiving (distinct from listening) and transmitting data; `radio_cca` provides an indication of channel state when the radio is listening (power enabled and not transmitting); and `radio_txready` indicates that a frame which has been queued is ready to be transmitted.

The interface to the on-board static RAM is directly available to the MAC subsystem, as is control of the two LEDs. The remaining MAC interfaces are 32-bit wide control and status interfaces which map to registers accessible from the PCI interface. These allow simple control of MAC operation such as on/off switching and reset, along with simple feedback on MAC and radio state.

6.1.3 MAC Subsystems

The framework described in Section 6.1.2 provides for specific MAC implementations to be 'dropped in' to allow rapid implementation. Full flexibility is available to the designer in allocating implementation of MAC elements between configurable hardware and embedded software. To further speed the development process for users, two base MAC sub-systems have been developed. These are described in the following sections.

Capture MAC

The capture MAC is named for its origins as a simple full-packet capture implementation. It has since been extended to provide transmission facilities as well. Figure 6.8 shows the body of the capture MAC architecture in VHDL. This serves to illustrate the ease with which a basic system can be implemented within the WAG framework.

Within the capture MAC, direct connections are made between the radio and host sides of the receive data stream, and the transmit data stream. The radio power enable and channel are connected directly to the MAC control register available from the PCI interface, and transmission of queued frames is per-

```
-- Aliases for bits within the MAC control register
alias control_pe      : std_logic is control(31);
alias control_channel : std_logic_vector(3 downto 0)
                          is control(3 downto 0);
signal led_txrx       : std_logic;
                   ...
-------------------------------------------------------
-- Radio power control: hold power enable until the end of
-- transmission or reception of a current packet
radio_pe <= control_pe or radio_transmitting or radio_receiving;
-- Start transmit when...
radio_txstart <= control_pe
                 and radio_txready
                 and not radio_transmitting
                 and not radio_receiving
                 and radio_cca;
-- Channel change mechanism
radio_channel <= control_channel;

-------------------------------------------------------
-- Transmit path.
wag_mac_rd      <= mac_radio_rd;
mac_radio_empty <= wag_mac_empty;
mac_radio_data  <= wag_mac_data;
-- Receive path.
mac_wag_data    <= radio_mac_data;
mac_wag_empty   <= radio_mac_empty;
radio_mac_rd    <= mac_wag_rd;

-------------------------------------------------------
-- LED 0 indicates the radio is on
led(0) <= control_pe;
-- LED 1 indicates radio activity (with 50ms POV)
led_txrx <= radio_transmitting or radio_receiving;
led1 : entity work.led_pov
  generic map (input_type => "activehigh",
               clk_freq => auxclk_freq, pov => 50000)
  port map (clk => auxclk, reset => reset,
            input => led_txrx, output => led(1));
```

Figure 6.8: Capture MAC Implementation

formed when the radio is power-enabled, purely listening, and the channel is deemed clear. Finally, the first LED is connected to the power-enable signal, and the second is tied to radio activity using a library entity (`led_pov`) to provide persistence-of-vision for aesthetic purposes. The `led_pov` entity is one of several VHDL entities available for use in MAC implementation. Others provide functionality such as signal resynchronisation, edge detection, and timers. Like the `led_pov` entity, these are parameterisable through the VHDL generic map mechanism.

The capture MAC provides a useful tool for tracing of wireless activity, and generation of arbitrary packets. Simple protocols can be implemented in software running on the host using the device driver described later in Section 7.2 and, using the capture MAC as a basis, more complex channel access mechanisms can readily be implemented in logic between the radio status and control signals. As the complexity of the MAC increases, however, a richer environment for development is necessary. This is provided for by the second of the example MAC subsystems that has been produced.

The PowerPC MAC Framework

The PowerPC MAC framework provides a subsystem which allows rapid implementation of protocols developed within the WPDE by using the PowerPC processor embedded within the FPGA for execution of code generated by the sdl2cpp tool described in Chapter 4.

The PowerPC core block has been incorporated in an architecture with custom peripherals to provide a system optimised for MAC protocol implementation. This architecture is illustrated in Figure 6.9. The instruction and data-side on-chip memory blocks (left-most in the figure) have dedicated connections to the processor to maximise bandwidth. The Processor Local Bus (PLB) provides a high bandwidth to crucial peripherals including the FIFO interfaces to and from the radio and host, and the General Purpose Input/Output (GPIO) ports. A bridge from the PLB provides access to the slower On-chip Peripheral Bus (OPB) which accommodates the external SRAM interface, a programmable

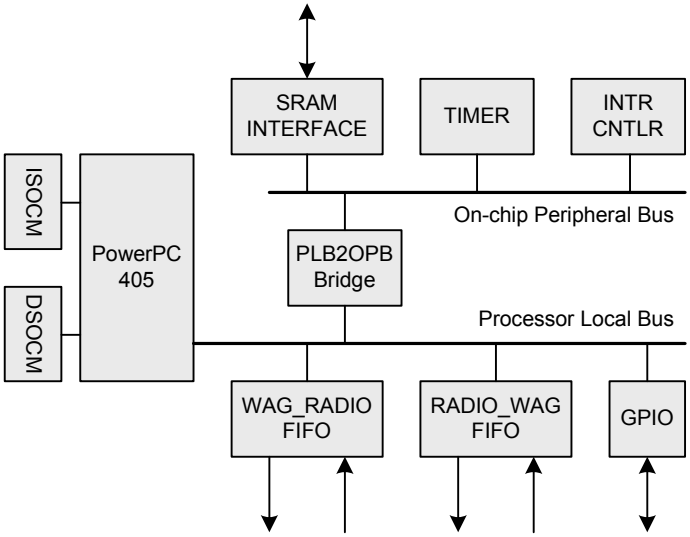


Figure 6.9: PowerPC Subsystem Architecture

timer, and the interrupt controller. The processor module supports caching of memory regions for increased performance, and this is typically applied to the external SRAM address range.

The embedded software that provides the API allowing integration of protocols translated using *sdl2cpp* is implemented in a combination of C and C++. A main loop polls the input sources and injects signals into the virtual SDL environment as appropriate. Data references are implemented as WAG frames for timing efficiency, though only the common frame header, payload length, and payload body fields are accessible through the data reference operators. A simple memory allocator is implemented to provide for management of packets, and operates in the memory area corresponding to the external SRAM.

Records are kept of all active SDL timers, and the single system timer is manipulated appropriately to ensure that it aligns with the next event. Use of a C++ class with static members ensures that this detail is hidden from the virtual SDL environment.

By default the radio status signals are mapped into the processor subsystem through the GPIO port, with the radio control signals similarly mapped as outputs from the processor. This is hidden from the MAC sub-layer in most circumstances, however this approach provides the flexibility to implement

advanced channel access mechanisms in either VHDL or C if desired.

6.2 Driver

To facilitate evaluation of protocol implementations in a representative environment, a Linux network interface driver for the WAG device has been developed[1]. This driver implements the interfaces to the Linux network stack and the PCI WAG device, and provides basic network functionality.

An attempt to transmit a packet is made through invocation of the driver routine `wagnet_start_xmit` by the kernel. A parameter of this call is the Linux socket buffer representing the data unit which should be sent. The data within this buffer is formatted within an 802.3-style header beginning with source and destination MAC addresses. The addresses and data are extracted and formed into a WAG frame corresponding to the MAC-UNITDATA.request primitive (introduced in Chapter 4), which is then passed through the host interface to the MAC. On invocation of the `wagnet_start_xmit` routine, the queue from the host is paused using the `netif_stop_queue` system call, and remains in this state until a MAC-UNITDATA.confirm is returned from the device, at which point the queue is made active again through a call to `netif_wake_queue`.

When a complete frame is available to the host, the `wagnet_rxinterrupt` driver routine is invoked. If the frame type is MAC-UNITDATA.confirm, then the transmit queue is un-blocked as described above. Alternatively, if the received frame is a MAC-UNITDATA.indication, then the payload data is extracted into a new socket buffer, which is passed to the host stack using the `netif_rx` call.

Framework is provided for implementation of custom `ioctl` hooks, allowing user-space programs on the host to control device or protocol-specific operation. A utility function allows manipulation of MLME attributes, and is

[1] A character driver for the WAG card is also provided. Though primarily intended for measurement purposes, this is often a useful tool for initial protocol testing. The character driver is discussed in more detail in Chapter 7.

used to provide support for MAC address configuration through the `ifconfig` interface.

6.3 Summary

The implementation platform described in this chapter enables a protocol design to be taken to implementation with relative ease. Various mixes of hardware and software implementation are supported, from full automatic implementation in embedded software using the sdl2cpp tool described in Chapter 4, through to optimised implementation of elements of MAC functionality using VHDL.

Though the realisation presented in this chapter uses an 802.11b physical layer designed for a CSMA/CA system, the architecture described could make use of any other technology desired. However, the choice of 802.11b PHY is not as restrictive as it may seem: TDMA protocols may be implemented readily using the configurable hardware to schedule transmissions accurately, and the timing of FHSS protocols such as Bluetooth can be readily evaluated; CDMA protocols may be approximated through use of specific 802.11 channels in the ISM band. Several modulation and coding options are available as part of the 802.11b PHY, and other coding rates may be achieved through their implementation in the digital logic of the FPGA.

The WAG device provides a powerful and portable platform which allows deployment of prototype implementations in their intended application environment. This hardware also provides support for the evaluation of these prototype protocols – a process which must now be discussed.

Chapter 7

Measurement

Measurement has a key role to play at multiple stages of the protocol design process. Evaluation of existing protocols leads to the development of new and improved ones, which in turn must be evaluated to ensure their suitability for the task. Simulation provides a flexible method of protocol evaluation in situations where the environment surrounding the protocol entity – including traffic and physical layer behaviour – is well understood. In certain cases this condition may not hold, and measurement provides a means for developing an understanding of the effect of external factors on protocol performance.

Numerous studies have used forms of measurement for evaluation of existing protocols [28, 34, 37, 102]. Such studies often identify weaknesses in the protocol being considered, and many propose modifications to the protocol or develop a new one to address the problem. As discussed in Chapter 6, however, few of these developed protocols progress to implementation, and hence few are thoroughly evaluated through measurement. To fully support the MAC protocol development process it is necessary to provide for measurement within the WPDE framework.

Network measurement encompasses many different techniques applied at various layers of the network stack. At the lowest layers of the network stack, measurement may involve analysis of the modulated signal to evaluate param-

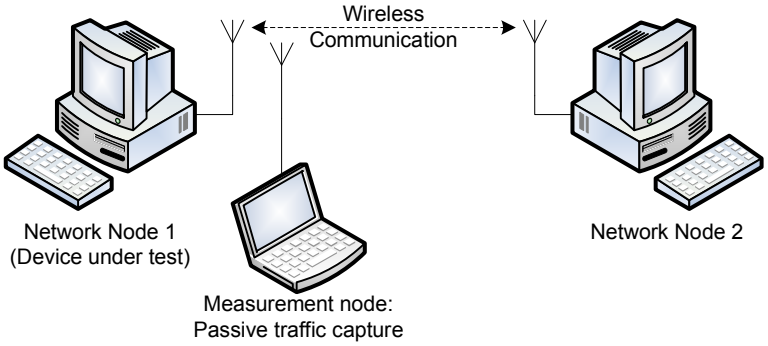


Figure 7.1: Indirect Wireless Packet Capture

eters such as modulation accuracy in terms of Error Vector Magnitude (EVM), or the transmission spectral mask. Conversely, higher layer measurements may involve characterisation of application performance when operating over the protocol under test. Network layer and above measurements are implicitly supported for protocols developed within the WPDE framework through the implementation platform presented in Chapter 6 which includes a network interface driver. Both of these extremes are important aspects of measurement, however, within the scope of this thesis it is the operation of the MAC sublayer and its interactions with its adjacent layers that is of primary concern. Passive event tracing provides an effective form of measurement at these levels.

At its simplest, the measurement output from passive event tracing is a temporally ordered sequence of event identifiers. If a reference time source is available, then the trace can be extended to be a sequence of tuples consisting of an event identifier, and a timestamp. In an implementation, this can remove the requirement for the sequence to be temporally ordered, as any reordering can be corrected in post-processing. A further extension is the addition of event information to the trace entry. This may include information like the full or partial contents of a received frame, or a snapshot of some device state at the event time. From an event trace it is trivial to generate a message sequence chart which provides an excellent visualisation of device operation. This technique is commonly applied at the network layer through tools such as tcpdump [103] or WireShark [104], and at the physical layer through tools such as OmniPeek [105].

Existing techniques for measurement at the physical layer interface typically involve capture of packets exchanged over the wireless medium by a third-party device. Wireless analysis systems such as OmniPeek use compatible wireless NICs coupled with a custom driver to place the card into a mode which allows capture of all received packets. This approach is referred to as *indirect capture*, and is depicted in Figure 7.1. Unfortunately, indirect capture does not necessarily lead to accurate measurement results.

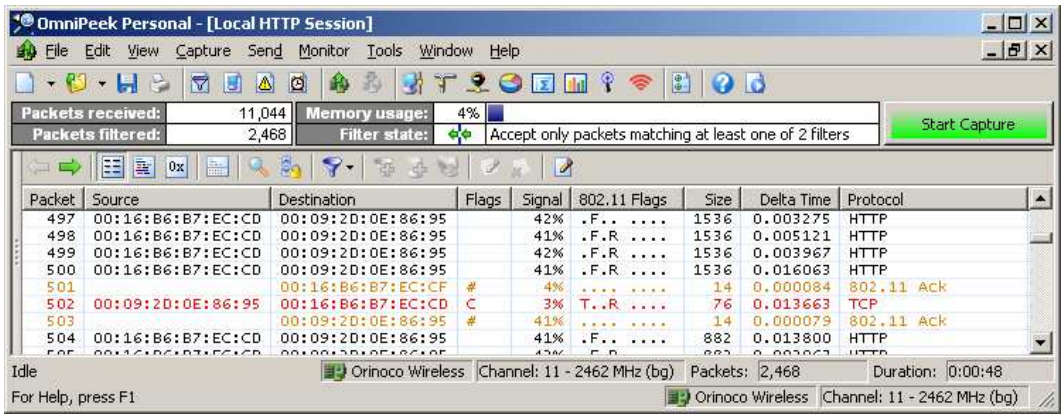


Figure 7.2: OmniPeek Capture of an 802.11b Link

Differences between the activity detected by the measurement node and that detected by the nodes participating in the network can lead to incomplete or misleading packet traces. In certain circumstances the actual packet exchanges may be extrapolated given knowledge of the protocol, however this is not always possible. Figure 7.2 illustrates the problem. Shown is an 802.11b capture taken using the OmniPeek tool. Packets 498, 499, and 500 show IEEE 802.11 MAC retransmissions of a data frame carrying HTTP traffic. Despite all these frames being received correctly by the monitor node, they were either received in error or not at all by their intended destination, or a returned acknowledgement was not heard either at the monitor or the originating station. Packet 502 illustrates an example of a frame which is received in error by the monitor node (indicated by the 'C' in the flags column showing a CRC error), but apparently received correctly by the intended destination (given that an acknowledgement frame is immediately returned).

The fundamental cause of the problem is that ability to receive a wireless signal correctly is often very location dependent. The effect of both passive

and active interferers can cause significant discrepancies between the RF signal as observed at two locations which may be physically close. Reducing the physical distance between the measurement receiver and the network node can improve the relevance of the measurement. This can be taken to the point of electrically coupling both radios to the same antenna. However, this still does not completely address the problem. Minor process variations between printed circuit boards, and the components that make up a receiver can lead to two supposedly identical devices having different receive characteristics. The effect of these variations is most noticeable in situations where the signal to noise ratio is marginal. Signal to noise ratio can be maximised by conducting tests in an RF shielded box lined with anechoic foam, however this is not necessarily compatible with the desire to measure systems operating in a realistic environment.

One approach to solving this problem is to integrate the network and measurement nodes. This ensures that the events captured within the measurement are consistent with the network node activity, though care must be taken to ensure that measurement activity does not interfere with network operation. In contrast to the indirect capture shown in Figure 7.1, this approach is herein referred to as *direct capture*.

Capture at the network interface is considerably easier than at the physical layer. Packets crossing this interface can be traced using the tcpdump tool [103], which provides a view of activity in both directions at the interface between the network driver and the host stack. Ethereal [106] and WireShark [104] are tools that provide similar facilities. These present a similar user interface to that of some wireless protocol sniffers, but must not be confused with physical layer tools as they provide only a view into activity at the lower boundary of the network layer. Figure 7.3 shows a screen-shot of a WireShark capture of the initial stages of a TCP Iperf [107] bandwidth test. It is relevant to note that although the test was conducted across an 802.11b wireless network, the captured trace gives no indication of this.

Network layer capture tools are useful for basic network troubleshooting, and can provide relative timing of network stack activity to a moderate level of

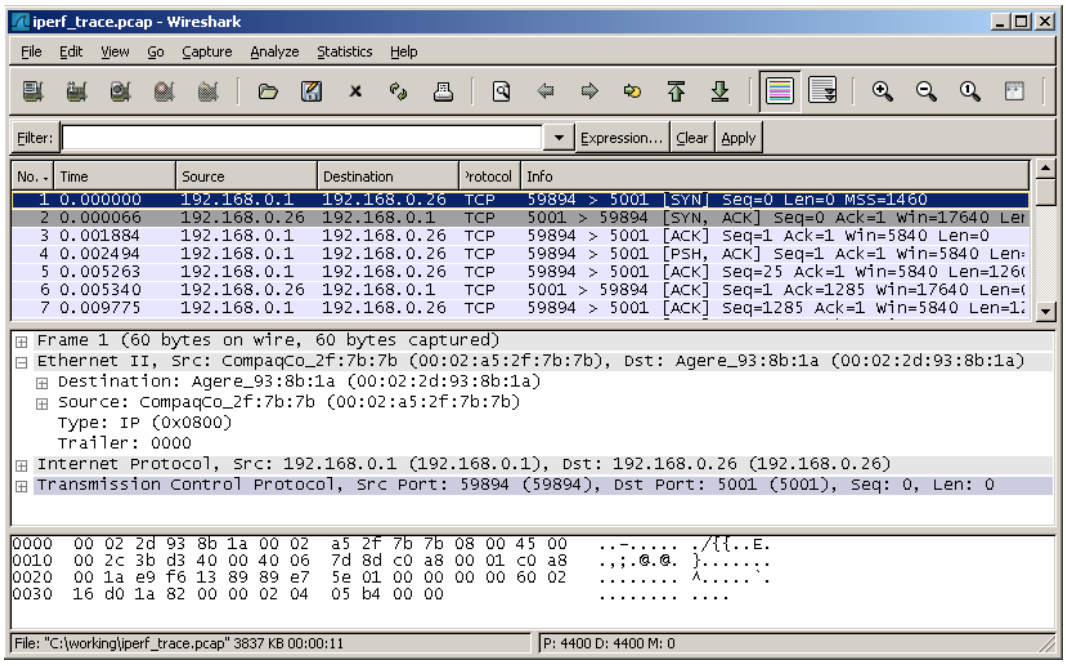


Figure 7.3: WireShark Capture of Traffic on a Wireless Network

accuracy. They have, however, two main shortcomings when used in protocol development. Firstly, they provide no information related to non-data host interface activity. Only packets that are passed between the device driver and the network stack will be visible. Although this is the primary information that is of interest, it is also useful to be able to view the associated flow control and MLME signalling. Secondly, these systems typically provide no means for accurately correlating the timing of the observed traffic to traces taken from other sources. This prevents, for example, accurate comparison between network layer activity and the resulting wireless communication. This information is vitally important in protocol development, as host interface performance can have a significant impact on overall network node performance.

Given measurement at the physical and network layer interfaces, a further aspect of interest is the relationship between observed activity on these, and changes in elements of internal MAC state. A microprocessor debug interface can provide a substantial amount of this information in troubleshooting. Typically however, these interfaces require execution to be paused before allowing

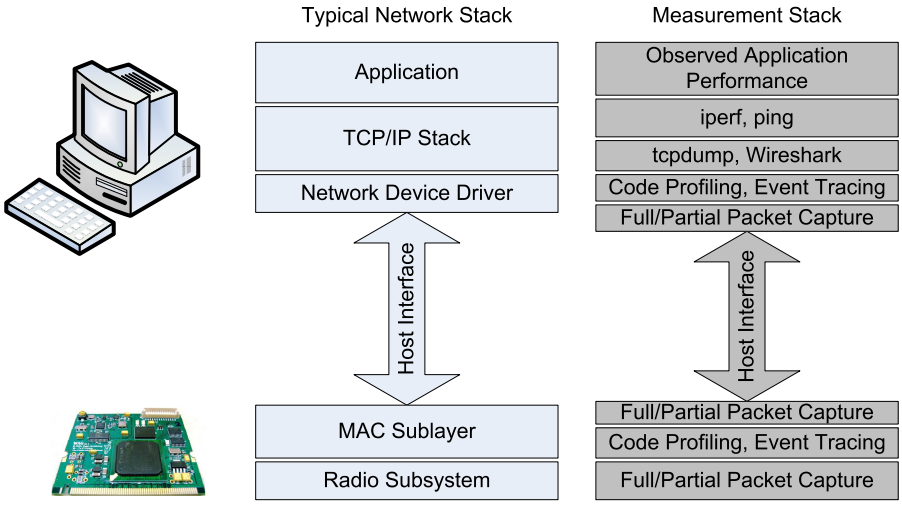


Figure 7.4: Integrated Measurement Stack

access to any useful processor state. They therefore will almost certainly impact on the operation of the device under test.

A second common method for tracing the state of a high-speed digital device is through the use of a logic analyser. This approach can provide a non-intrusive means for observing some internal state with a very high degree of timing accuracy. Unfortunately the amount of state that can be viewed is typically limited, either by the generally low number of available debug GPIO pins, or the number of channels available on the logic analyser. This has the follow-on effect of making it difficult to correlate observed state changes, with other measured activity, such as captured radio transmissions. A second key consideration is that logic analysers are typically expensive, and require a degree of specialist knowledge that can be outside the scope of a network protocol engineer's skills.

Ideally, the WPDE framework will provide for tracing of internal MAC state. In Chapter 4 a measurement service primitive was defined to allow the MAC to indicate key state changes for use in the debugging and analysis of the protocol. This primitive should be integrated within the measurement framework which the remainder of this chapter defines.

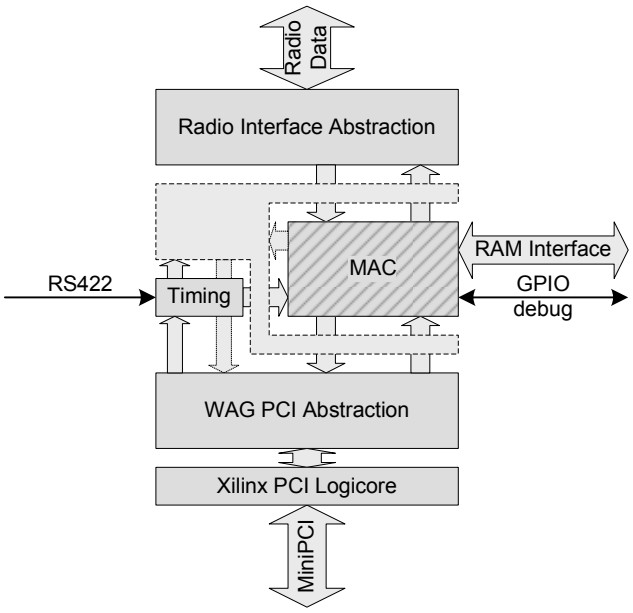


Figure 7.5: WAG Measurement Architecture

Figure 7.4 gives an overview of the layers in a practical network stack, and the measurement techniques that we may wish to apply at each level. Within the WPDE framework it is necessary to support packet or event tracing within the MAC sub-layer, and at the interfaces around it. In particular the framework must support non-invasive capture of physical layer activity in an operational network node. The ability to correlate measurements taken at these various measurement points is also beneficial, so time-stamping of events should use a consistent time reference.

7.1 WAG Measurement Architecture

To support measurement within the WPDE, the WAG implementation framework has been extended as shown in Figure 7.5. The additions provide for passive capture of activity on the interfaces surrounding the MAC entity, including the dedicated measurement service access point. The timing and synchronisation unit provides a common time reference to allow correlation of

measurements taken at the various points, and enables synchronised measurement across physically distributed nodes.

A significant decision made in the design of the architecture of Figure 7.5, was the decision to multiplex measurement data over the existing PCI network host interface. Though a dedicated channel for carrying measurement data off-board would have ensured that measurement traffic did not interfere with network operation, two key considerations were used to justify this design choice:

Portability: A design requirement was to support implementation in relatively remote environments (i.e. not in a laboratory). The targeted host platform for such environments was a single-board computer with limited capability for expansion. Use of the host interface allows for measurement capability in this situation.

Bandwidth: The bandwidth requirements of the chosen physical layer for the WAG device (802.11b at 11Mbps maximum serialisation) were sufficiently less than the available host interface bandwidth (Mini-PCI: 32 bit at 33MHz = 1056Mbps maximum), meaning that contention for bus resources would likely be minimal.

The basis of the measurement support in the WAG framework is a number of custom hardware blocks designed in VHDL for implementation in the FPGA. These blocks are discussed in the following sections.

7.1.1 Measurement Channels

A key part of the measurement framework is the channel which conveys measurement data to the host system. Though this may seem a simple task, it is important to minimise the observer effect of the measurement on the operation of the network device. The observer effect refers to changes in the behaviour of a system under test due to supposedly non-intrusive observation. In designing a system with integrated measurement, it is important to min-

imise the observer effect on the device under test. Although digital signals typically have sufficient entropy to prevent observation affecting their validity (presuming driver loading is not excessive), higher level architecture design must include consideration of this phenomenon. This is especially pertinent given the multiplexed measurement and network data approach that is to be taken for the host interface. Three distinct blocks are defined to control the flow of measurement data and manage the observer effect: a multiplexer, a de-multiplexer, and a duplicator.

The multiplexer block allows combination of two WAG frame FIFO streams. This is achieved by interleaving from the input streams into the output in units of entire frames. When a frame header becomes present on one of the multiplexer inputs, it takes a copy of the frame size field into its transfer counter. The output stream is then held by this frame, and the frame data words are transferred on agreement between the source and destination, with the transfer counter decrementing on each transfer. When the transfer counter reaches zero, the multiplexer returns to its idle state. A key concern in the use of such a block is the potential for measurement frames to impact on the transit of network frames. In the case where both inputs have frames ready-to-go, a decision must be made as to which one to take. The multiplexer block allows priority to be assigned to the network frame source over the measurement frame source to minimise the observer effect.

The de-multiplexer block provides the opposite function to the multiplexer by splitting a single frame stream into two. This block operates in a similar fashion to the multiplexer, with the frame type field in the header used to determine the destination of each frame received on the input port. If one of the de-multiplexer output streams is blocked in output, then any frames behind that one will be delayed. For this reason it is vital that measurement streams be drained at their ultimate destination as quickly as possible.

The duplication block performs a similar function to the de-multiplexer in splitting a single stream into two, however instead of interleaving frames between the two output streams, each input frame is replicated on both. Stream duplication presents similar challenges to multiplexing and de-multiplexing in

terms of avoiding the observer effect. If frames are to be duplicated whole or not at all, and the duplication is not to have any effect on the main path, then the duplicator block must provide enough buffering for the largest frame it will see.

Though undesirable, blocking of streams may be unavoidable occasionally in a system with finite resources. Such blocking may cause measurement frames to be lost, leaving potential for confusing measurement results. In cases where this occurs it is important that the loss is indicated within an event trace. The WAG frame header defined in Chapter 6 provides for this through the loss counter field.

7.1.2 Timing

The timing block is responsible for providing an accurate time reference to allow correlation of the discrete measurements taken throughout the system. This block provides a constantly increasing counter which is sampled at the instant of an event to provide a timestamp for that event.

The required granularity of time-stamping for measurements in such a system is dependent on the aspect of interest. In some circumstances event order may be sufficient to glean the necessary information. If timing information is required, then various parameters within the system may have impact on the necessary accuracy, including the serialisation rate of the specific physical layer, the transfer speeds of the host interface, MAC sub-layer data paths, and the capabilities of the MAC processor. With sub-microsecond accuracy it becomes possible to measure signal propagation delays over typical wireless network distances, though this requires some timing synchronisation between measurement nodes to the same level of accuracy. There is no drawback to providing a higher degree of timing accuracy than is necessary, and so in the WAG framework the resolution is constrained by the system clock (100MHz).

In the implementation, a counter on the digital system clock provides the basis for the measurement reference time. This block provides 64 bits of output,

using a fixed-point representation where the number represented by the most significant 32 bits has a unit of seconds, and the lower 32 bits provide the sub-second resolution where a count corresponds to $\frac{1}{2^{32}}th$ of a second. The implementation of this counter uses a 96 bit counter for increased accuracy. On each cycle, a value known as *count_per_cycle* is added to the counter. This value is initially set appropriately such that the lower 64 bits of the counter will wrap each second.

Optionally, a pulse-per-second input can be used to manage clock drift and provide synchronised timing across multiple (possibly distributed) WAG nodes. The designed hardware provides a phase locked loop on this pulse by resynchronising to it and automatically adjusting the value of *count_per_cycle* dependent on the determined error. Finally, the upper 64 bits of the counter can be manually loaded with a value through the host interface. Using the Network Time Protocol (NTP) or similar, the host can set the counter to within $\pm 0.5s$ of real time, and allow the pulse-per-second input to complete the synchronisation.

The event detection blocks described in Chapter 6 are also used for the measurement event detection and time-stamping blocks. These are extended to provide latching of the reference time provided by the timing unit on occurrence of the specified event. This timestamp block provides a portable source of timestamps which can be triggered by an event on any signal within the MAC framework.

Instantiations of these blocks are present in the receive and transmit units to provide accurate time-stamping of physical layer events. Triggering of such events is taken as close as the hardware will allow to the appearance of the first MPDU symbol on the air.

7.1.3 Frame Generation

As previously discussed in Chapter 6, the basic unit of data transfer within the hardware is known as a WAG frame. The existing WAG framework provides for transport of data which complies with the WAG frame format over the PCI

interface to the driver. In Section 7.1.1 blocks have been defined to allow the multiplexing of measurement data along with network data over this existing transport. To utilise this framework we define a frame format for carrying measurement information.

Each measurement frame should represent a discrete event within our framework. This may include the reception of a packet by the radio, the arrival of a packet at the MAC sub-layer over the network interface, or simply a change in internal MAC state. The fundamental information required within a measurement frame is the timestamp, and some means of identifying the event. To support packet capture the frame should also support the attachment of arbitrary data.

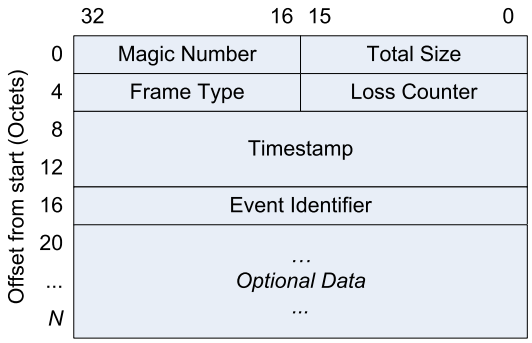


Figure 7.6: Basic Measurement Frame

The basic WAG measurement frame follows the format shown in Figure 7.6. Along with the standard frame header consisting of magic number, frame size, type and loss counter fields, the measurement frame also provides for inclusion of a timestamp, event identifier, and optional data.

The provided measurement frame generation block is a triggered source of measurement frames. It incorporates the event detection block to provide timestamping facilities, and will generate a WAG measurement frame including the timestamp, and the appropriate event identifier which is specified as a parameter at block instantiation. This block provides a WAG frame FIFO output as described in Chapter 6, allowing easy incorporation with the stream flow control blocks described in Section 7.1.1.

7.1.4 Firmware Support

The MAC interface defined in Chapter 4 includes a measurement primitive to allow indication of key changes in internal MAC state. Two approaches are possible to the generation of measurement frames based on the output of this signal. The MAC firmware may generate measurement frames and pass these back to the host through its normal channel, though care should be taken to ensure the extra processing required will not invalidate the intended measurement. In this case the firmware is responsible for setting the timestamp value. To support this approach the timestamp counter is made available to the PowerPC processor through a memory-mapped register.

An alternative approach is to use the provided dedicated measurement event interface which attaches to the PowerPC framework. To firmware this interface provides a register, which – when written – will cause a timestamp to be taken, and a measurement frame to be generated with the written value as the event identifier. This is the preferred approach due to the reduced processing burden on the firmware and accurate timestamps provided through the dedicated hardware support.

7.2 Host Tools

The ultimate de-multiplexing of network and measurement frames is performed in the device driver. The Linux driver described in Chapter 6 provides both a network interface to which all network frame payloads are directed, and a character device which is intended for measurement trace capture. The structure of the driver is shown in Figure 7.7. The network aspects of the driver and the architecture of the host interface have been discussed in Chapter 6. Here we consider the features that pertain to integrated measurement.

The PCI abstraction layer within the driver is responsible for separating the network and character driver subsystems from the specific detail of the WAG memory-mapped interface. It is also responsible for the distribution of frames

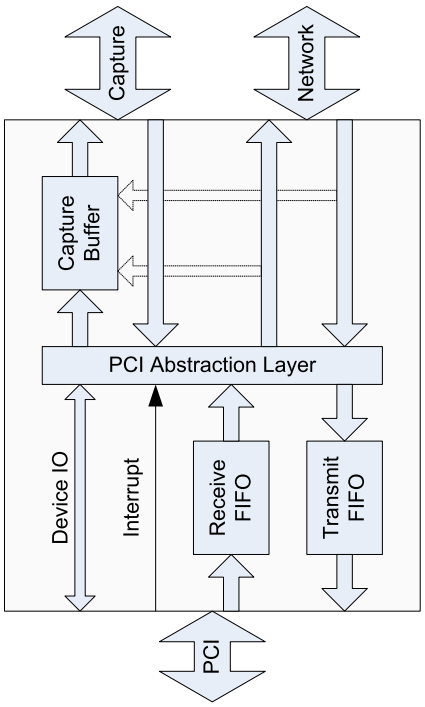


Figure 7.7: WAG Device Driver Architecture

received over the host interface. An interrupt from the WAG to the host indicates the transfer of some amount of data either to the receive, or from the transmit FIFOs. If the driver interrupt handler determines that a complete frame is available in the receive FIFO, then it will determine the destination of that frame from its type and subtype fields and push it to the appropriate subsystem.

As discussed earlier, it is vital that the measurement stream be drained as quickly as possible. To allow this, the capture subsystem provides a buffer in the card-to-host direction, and if this buffer becomes full then any frame destined for capture will be dropped to allow the progression of network data. A counter tracks the number of dropped frames and updates the loss counter field of the next successfully captured frame to ensure the user is aware of the loss.

Optionally, all network frames received or sent over the host interface can be

copied to the capture device as well. This allows a simple view of host interface activity without any measurement enabled on the WAG card.

Access to the transmit FIFOs is shared (using a semaphore for locking) between the two driver subsystems. This allows for the generation of frames for device testing and control through relatively simple user-space programs.

The Linux character device model means that access to the capture device presented by the driver is relatively simple. The device can be opened as a file (`/dev/wag`) and raw frames are read from, or written to this descriptor.

Both frame capture and generation utilities are provided, along with a tool for displaying captured traces in a human-readable form. These tools provide the basic elements required for protocol measurement.

7.3 Summary

The measurement framework described in this chapter provides a flexible toolkit for the evaluation of protocols implemented within the WPDE. The architecture presented addresses the problems of indirect capture and provides for synchronised measurement at the physical and network interfaces, which can be correlated with changes in internal MAC state. Sufficient timing resolution is provided to allow detailed physical layer measurements including packet transit times, and the use of GPS technology allows highly synchronised, physically distributed measurements to be taken.

The software and hardware elements of the measurement system support the protocol designer by providing for rapid construction of debugging and evaluation framework for a custom protocol implementation. This allows protocols to be assessed in a real-world environment to ensure a representative view of their performance.

Chapter 8

Conclusions

Wireless networking is an active and important area of research, and Medium Access Control (MAC) protocol development forms an integral part of this. An increasing amount of functionality is being pushed into the domain of the MAC protocol, and yet its position just above the physical layer requires that strict timing constraints are met in its low level functions. These conflicting demands require a disciplined approach to MAC protocol design.

A MAC can be considered as a real-time distributed algorithm with high reliability requirements. It is generally accepted that use of formal methods in such systems is best design practice for ensuring a robust end-product. The Specification and Description Language (SDL) provides an intuitive, graphical design environment for network protocol designers.

By defining a common interface for MAC protocols developed within the framework, automation can be used to ensure consistency between the initial specification, and simulation models or implementations. A tool is described, which is able to convert SDL protocol descriptions into efficient C++ code implementations for integration into the target simulation and implementation platforms.

Simulations at the various levels of abstraction provide feedback appropriate for the current design stage. Direct simulation of the SDL specification gives a first view of its operation, and provides feedback on protocol correctness.

Simple traffic and channel models have been defined to support this process within a general purpose SDL simulation environment. The level of detail in traffic and physical layer models is shown to be an important consideration if conclusions on protocol performance are to be drawn. A described framework for automated integration of designed protocols into the *ns-2* simulator allows access to its thorough range of simulation models and emulation facilities.

Prototype implementation provides the best opportunity of estimating final system performance. This allows thorough performance evaluation of the system in a 'real-world' scenario. A hardware, firmware, and software framework for automated implementation of wireless MAC protocols under development is described, allowing implementation to be used at an early stage in the design process. The hardware provides a MAC-less radio with an 802.11b physical layer in a mini-PCI form factor. It includes an FPGA which incorporates a PowerPC processor, allowing for flexibility in design implementation in either hardware or software. The device driver framework for Linux allows ready implementation of a network device driver for implementation, as well as separation of network and measurement traffic from the device under test.

The described measurement framework allows synchronised and distributed passive measurement across nodes, with direct capture of physical layer activity. Measurement points integrated in an operating network node mean that network and physical layer activity and MAC state can be related to allow debugging and a better understanding of protocol operation. The hardware allows for real-time synchronisation of distributed measurement nodes by way of GPS pulse-per-second output and NTP. Measurement units can be inserted in the MAC data-path hardware, and configured to trigger on various internal conditions.

8.1 Case Studies

The development of the PTPMAC protocol has demonstrated the use of all components of the described framework, from requirements capture and initial

specification in SDL, through SDL and *ns-2* simulation to implementation stages.

The WPDE framework has been used in other studies involving wireless medium access control protocols. Three of these are briefly discussed here in order to illustrate the way in which the framework facilitates rapid design of wireless MAC protocols.

Wireless Location Determination

An investigation by Bartels [108] used elements of the WPDE framework to construct a location determination system. This used the accurate time-stamping provided by the measurement elements of the framework to measure time-of-flight for packets, and thus derive a measure of the physical separation of nodes. A simple protocol to support the necessary exchange of information was developed using the WPDE host software and driver, and the extended capture MAC framework. Planned further work aims to integrate time-of-flight measurement into an extension to the IEEE 802.11 protocol.

Wireless MAC Protocols for Rural Environments

Recently started work is seeking to develop wireless MAC protocols for use in rural environments. The WPDE platform has supported the early stages of this work in which a Point-to-Multi-Point MAC protocol has been developed using the design flow described within this thesis. This protocol uses a dynamic TDMA method controlled by the base station, and has been taken rapidly from initial concept to a working prototype implementation using the WPDE framework.

Wireless Mobile and Ad-Hoc MAC Protocols

This work involves the design of a contention-based MAC protocol which will incorporate location estimation and neighbour mobility prediction features. This will allow the exploration of cross-layer interactions between MAC and advanced routing protocols. Initial elements of MAC functionality have been implemented in VHDL within a derivation of the capture MAC, and the WPDE framework will also provide support for simulation of the protocol.

8.2 Summary

The dearth of frameworks for supporting wireless MAC protocol research has likely restricted progress in this area. Such protocols are complex distributed systems which require careful design processes. This thesis concludes that the wireless MAC protocol development framework described herein – the Waikato Protocol Development Environment – facilitates the advancement of knowledge in the field of wireless MAC protocol research by supporting a rapid transition from protocol design concept to prototype implementation, encouraging sound design methodology throughout. Consideration is being given to the various options for making the described framework available to researchers. Further information on this will be available through the website of the WAND Network Research Group [109].

Bibliography

[1] Norman Abramson. The Aloha System – Another Alternative for Computer Communications. In *Proc. AFIPS Conference*, volume 36, pages 295–298, 1970.

[2] IEEE Computer Society. *IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, 1999.

[3] Wi-Fi Alliance, [Online]. Available: http://www.wi-fi.org/. [Accessed Oct. 30 2006].

[4] Bluetooth SIG. *Specification of the Bluetooth System, version 1.2*, November 2003.

[5] IEEE Computer Society. *IEEE 802.16: Air Interface for Fixed Broadband Wireless Access Systems*, 2004.

[6] WiMAX Forum, [Online]. Available: http://www.wimaxforum.org/. [Accessed Aug. 31 2006].

[7] ETSI. *Broadband Radio Access Networks; HiperMAN; Data Link Control (DLC) Layer*, March 2006.

[8] WiBro Personal Internet Service, [Online]. Available: http://www.wibro.or.kr/. [Accessed Oct. 30 2006].

[9] ETSI. *Universal Mobile Telecommunications System (UMTS); Medium Access Control (MAC) protocol specification*, September 2005.

[10] ETSI. *Universal Mobile Telecommunications System (UMTS); UTRA High Speed Downlink Packet Access (HSDPA); Overall description*, December 2004.

[11] Matthew B. Shoemake. Wi-Fi (IEEE 802.11b) and Bluetooth Coexistence Issues and Solutions for the 2.4 GHz ISM Band. Technical report, Texas Instruments, February 2001.

[12] Wi-Fi (802.11b) and Bluetooth: An Examination of Coexistence Approaches. Mobilian Corporation, 2001.

[13] Falk Dietrich and Jean-Pierre Hubaux. Formal methods for communication services: meeting the industry expectations. *Comput. Networks*, 38(1):99–120, 2002.

[14] Axel van Lamsweerde. Formal specification: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 147–159, New York, NY, USA, 2000. ACM Press.

[15] Jean-Marc Jezequel Alain Le Guennec and Franois Pennaneac'h. Validating distributed software modeled with UML. In *Proc. UML'98 International Workshop*, pages 331–340, Mulhouse, France, Hune 1998.

[16] Andrei Gurtov and Sally Floyd. Modeling wireless links for transport protocols. *SIGCOMM Comput. Commun. Rev.*, 34(2):85–96, 2004.

[17] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *MSWiM '04: Proc. 7th ACM intl. symposium on modeling, analysis and simulation of wireless and mobile systems*, pages 78–82, New York, NY, USA, 2004. ACM Press.

[18] Sally Floyd and Eddie Kohler. Internet research needs better models. *SIGCOMM Comput. Commun. Rev.*, 33(1):29–34, 2003.

[19] Norman Abramson. The throughput of packet broadcasting channels. *IEEE Transactions on Communications*, 25(1):117–128, January 1977.

[20] Andras Varga. The OMNeT++ discrete event simulation system. In *Proc. European Simulation Multiconference (ESM'01)*, Prague, Czech Republic, June 2001.

[21] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Rajive Bagrodia, and Mario Gerla. GloMoSim: A Scalable Network Simulation Environment. Technical Report 990027, UCLA, Computer Science Department, May 1999.

[22] George F. Riley. Large-scale network simulations with GTNetS. In *Proceedings of the 2003 Winter Simulation Conference*, volume 1, pages 676–684, 2003.

[23] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. MANET Simulation Studies: The Incredibles. *Mobile Computing and Communications Review*, 9(4):50–61, October 2005.

[24] Kevin Fall and Kannan Varadhan. *The ns Manual*. The VINT Project, November 2005.

[25] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, 2000.

[26] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. Technical Report 99-702, University of Southern California, Los Angeles, 1999.

[27] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM Press.

[28] Vijaynarayanan Subramanian, K.K. Ramakrishnan, Shivkumar Kalyanaraman, and Lusheng Ji. Impact of interference and capture effects in 802.11 wireless networks on TCP. In *Proc. of Second International workshop on Wireless Traffic Measurements and Modeling*, Boston, MA, USA, August 2006.

[29] Gavin Holland, Nitin Vaidya, and Paramvir Bahl. A rate-adaptive MAC protocol for multi-hop wireless networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 236–251, New York, NY, USA, 2001. ACM Press.

[30] Chane L. Fullmer and J. J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 39–49, New York, NY, USA, 1997. ACM Press.

[31] Thanasis Korakis, Gentian Jakllari, and Leandros Tassiulas. A MAC protocol for full exploitation of directional antennas in ad-hoc wireless networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing*, pages 98–107, New York, NY, USA, 2003. ACM Press.

[32] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *USENIX Winter*, pages 259–270, 1993.

[33] David Kotz, Calvin Newport, and Chip Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth College, Computer Science, Hanover, NH, July 2003.

[34] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. In *Proc. SIGCOMM'04: Conference on applications, technologies, architectures, and protocols for computer communications*, pages 121–132, 2004.

[35] Kameswari Chebrolu, Bhaskaran Raman, and Sayandeep Sen. Long-distance 802.11b links: Performance measurements and experience. In *Proc. 12th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, Los Angeles, USA, September 2006.

[36] James Gross and Andreas Willig. Measurements of a wireless link in different RF-isolated environments. In *Proc. European Wireless '02*, pages 762–769, February 2002.

[37] Andreas Willig, Martin Kubisch, Christian Hoene, and Adam Wolisz. Measurements of a Wireless Link in an Industrial Environment Using an IEEE 802.11-Compliant Physical Layer. *IEEE Transactions on Industrial Electronics*, 49(6):1265–1282, December 2002.

[38] Aura Ganz, Andreas Savvides, and Zvi Ganz. Media access control development platform for wireless LANs. In *Proc. IEEE ICECS'99*, pages 105–108, 1999.

[39] Ananth Rao and Ion Stoica. An overlay MAC layer for 802.11 networks. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 135–148, New York, NY, USA, 2005. ACM Press.

[40] Bhaskaran Raman and Kameswari Chebrolu. Design and evaluation of a new MAC protocol for long-distance 802.11 mesh networks. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 156–169, New York, NY, USA, 2005. ACM Press.

[41] MadWifi Project Homepage, [Online]. Available: http://www.madwifi.org/. [Accessed Aug. 31 2006].

[42] Mathieu Lacage, Mohammad Hossein Manshaei, and Thierry Turletti. IEEE 802.11 rate adaptation: a practical approach. In *MSWiM '04: Proc. 7th ACM intl. symposium on modeling, analysis and simulation of wireless and mobile systems*, pages 126–134, New York, NY, USA, 2004. ACM Press.

[43] Chandrakanth Chereddi, Pradeep Kyasanur, and Nitin H. Vaidya. Design and implementation of a multi-channel multi-interface network. In *REALMAN '06: Proc. 2nd intl. workshop on multi-hop ad hoc networks: from theory to reality*, pages 23–30, New York, NY, USA, 2006. ACM Press.

[44] Maxim Raya, Jean-Pierre Hubaux, and Imad Aad. DOMINO: a system to detect greedy behavior in IEEE 802.11 hotspots. In *MobiSys '04: Proc. 2nd intl. conference on mobile systems, applications, and services*, pages 84–97, New York, NY, USA, 2004. ACM Press.

[45] Sourav Pal, Sumantra Kundu, Kalyan Basu, and Sajal Das. Performance evaluation of IEEE 802.11 multi-rate control algorithms using heteregenous traffic and real hardware. In *PAM '06: Passive and Active Measurement Conference*, Adelaide, Australia, March 2006.

[46] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1664–1669, 2005.

[47] TUTWLAN Research Homepage, Tampere University of Technology, Institute of Digital and Computer Systems, [Online]. Available: http://www.tkt.cs.tut.fi/research/daci/ra_tutwlan_overview.html. [Accessed Oct. 30 2006], 2006.

[48] Marko Hannikainen, Jarno Knuutila, Timo Hamalainen, and Jukka Saarinen. Using SDL for Implementing a Wireless Medium Access Control Protocol. In *Proc. Intl. Conference on Microelectronic Systems Education*, page 229, Washington, DC, USA, 2000. IEEE Computer Society.

[49] Marjo Kari, Marko Hannikainen, Timo Hamalainen, Jarno Knuutila, and Jukka Saarinen. Configurable platform for a wireless multimedia local area network. In *Proc. Fifth IEEE International Workshop on Mobile Multimedia Communications (MoMuc'98)*, pages 301–306, Berlin, Germany, October 1998.

[50] Kimmo Tikkanen, Marko Hannikainen, Timo Hamalainen, and Jukka Saarinen. Advanced prototype platform for a wireless multimedia local area network. In *Proc. European Signal Processing Conference (EUSIPCO'2000)*, pages 2309–2312, Tampere, Finland, September 2000.

[51] John Heidemann, Nirupama Bulusu, Jeremy Elson, Chalermek Intanagonwiwat, Kun chan Lan, Ya Xu, Wei Ye, Deborah Estrin, and Ramesh Govindan. Effects of detail in wireless network simulation. In *SCS Multiconference on Distributed Simulation*, pages 3–11, January 2001.

[52] Vassilis Tsaoussidis and Ibrahim Matta. Open issues on TCP for mobile computing. Technical report, Boston, MA, USA, 2001.

[53] David B. Johnson. Validation of wireless and mobile network models and simulation. In *Proceedings of the DARPA/NIST Network Simulation Validation Workshop*, May 1999.

[54] Elizabeth M. Royer, Sung-Ju Lee, and Charles E. Perkins. The Effects of MAC Protocols on Ad hoc Network Communications. In *Proc. IEEE Wireless Communications and Networking*, Chicago, IL, September 2000.

[55] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[56] IEEE Computer Society. *IEEE 802.11b: Higher-Speed Physical Layer Extension In the 2.4GHz Band*, 1999.

[57] IEEE Computer Society. *IEEE 802.11a: High-Speed Physical Layer in the 5GHz Band*, 1999.

[58] IEEE Computer Society. *IEEE 802.11j: 4.9GHz-5GHz Operation in Japan*, 2004.

[59] IEEE Computer Society. *IEEE 802.11g: Further Higher Data Rate Extension In the 2.4GHz Band*, 2003.

[60] Phil Karn. MACA - a new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, 1990.

[61] Vaduvur Bharghavan, Alan J. Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LAN's. In *Proc. SIGCOMM'94: Conference on Communications architectures, protocols and applications*, pages 212–225, 1994.

[62] IEEE Computer Society. *IEEE 802.11e: Medium Access Control (MAC) Quality of Service Enhancements*, 2005.

[63] IEEE Computer Society. *IEEE 802.11i: Medium Access Control (MAC) Security Enhancements*, 2004.

[64] IEEE Computer Society. *IEEE 802.16e: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands*, 2005.

[65] Doug Gray. Mobile WiMAX - Part I: A Technical Overview and Performance Evaluation. Technical report, WiMAX Forum, August 2006.

[66] Anthony Ira Wasserman. A top-down view of software engineering. *SIGSOFT Softw. Eng. Notes*, 1(1):8–14, 1976.

[67] Niklaus Wirth. Program development by stepwise refinement. *Commun. ACM*, 14(4):221–227, 1971.

[68] Randy H. Katz. *Contemporary Logic Design*. Benjamin Cummings/Addison Wesley Publishing Company, 1993.

[69] Marten van Sinderen, Luis Ferreira Pires, and Chris A. Vissers. Protocol design and implementation using formal methods. *The Computer Journal*, 35(5):478–491, 1992.

[70] Daniel D. Gajski and Frank Vahid. Specification and design of embedded software-hardware systems. *IEEE Design and Test of Computers*, 12(1), Spring 1995.

[71] Annette Bunker, Ganesh Gopalakrishnan, and Sally A. McKee. Formal Hardware Specification Languages for Protocol Compliance Verification. *ACM Transactions Design Automation of Electronic Systems*, 9(1):1–32, January 2004.

[72] Mark A. Ardis, John A. Chaves, Lalita Jategaonkar Jagadeesan, Peter Mataga, Carlos Puchol, Mark G. Staskauskas, and James Von Olnhausen. A framework for evaluating specification methods for reactive systems: experience report. In *ICSE '95: Proceedings of the 17th international conference on Software engineering*, pages 159–168, New York, NY, USA, 1995. ACM Press.

[73] Final Committee Draft on Enhancements to LOTOS. Technical Report WI 1.21.20.2.3, ISO/IEC, May 1998.

[74] Bernard Stepien and Luigi Logrippo. Graphic visualization and animation of LOTOS execution traces. *Comput. Networks*, 40(5):665–681, 2002.

[75] Hubert Garavel, Frederic Lang, and Radu Mateescu. An overview of CADP 2001. *European Association for Software Science and Technollogy (EASST) Newsletter*, 4:13–24, August 2002.

[76] TRAIAN: A Compiler for E-LOTOS Specifications, [Online]. Available: http://www.inrialpes.fr/vasy/traian/. [Accessed Nov. 20 2006].

[77] Francisco Isidro Massetto, Wanderley Lopes de Souza, and Sergio Donizetti Zorzo. Simulator for E-LOTOS Specifications. In *SS '02: Proceedings of the 35th Annual Simulation Symposium*, page 389, Washington, DC, USA, 2002. IEEE Computer Society.

[78] Justin George Templemore-Finlayson. A Graphical Representation For The Formal Description Technique Estelle. Master's thesis, The University of Cape Town, October 1998.

[79] Justin Templemore-Finlayson, Pieter S. Kritzinger, Jean-Luc Raffy, and Stanislaw Budkowski. A graphical representation and prototype editor for the Formal Description Technique Estelle. In *FORTE XI /*

PSTV XVIII '98: Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII), pages 37–55, Deventer, The Netherlands, 1998. Kluwer, B.V.

[80] Stan Budkowski. Estelle development toolset (EDT). *Comput. Netw. ISDN Syst.*, 25(1):63–82, 1992.

[81] ITU-T. *Formal description techniques (FDT) – Specification and Description Language (SDL)*, August 2002.

[82] SanDriLa Homepage, [Online]. Available: http://www.sandrila.co.uk/. [Accessed Nov. 23 2006].

[83] Telelogic TAU SDL Product Homepage, [Online]. Available: http://www.telelogic.com/products/tau/sdl/. [Accessed Oct. 30 2006].

[84] Cinderella Software Homepage, [Online]. Available: http://www.cinderella.dk/. [Accessed Oct. 30 2006].

[85] The CMU Monarch Project. The CMU Monarch Project's Wireless and Mobility Extensions to NS. Technical report, Carnegie Mellon University, August 1998.

[86] *IEEE 802.2: Standard for Information Technology Telecommunications and Information Exchange Between Systems - Part 2: Logical Link Control*, 1998.

[87] Ralf Henke, Hartmut Konig, and Andreas Mitschele-Thiel. Derivation of efficient implementations from SDL specifications employing data referencing, integrated packet framing and activity threads. In *Proc. 8th SDL Forum, SDL'97*, pages 397–414, Evry, France, Sept.

[88] Richard Sanders. Implementing from SDL. *Telektronikk*, (4), April 2000.

[89] Karl-Andre Skevik, Thomas Plagemann, and Vera Goebel Pal Halvorsen. Evaluation of a zero-copy protocol implementation. In *Proc. 27th Euromicro Conference*, 2001.

[90] Daniel Dietterle, Jerzy Ryman, Kai F. Dombrowski, and Rolf Kraemer. Mapping of High-Level SDL Models to Efficient Implementations for TinyOS. In *Proc. Euromicro Symposium on Digital Systems Design (DSD), Architectures, Methods and Tools*, pages 402–406, 2004.

[91] Michael Schmitt II. The Development of a Parser for SDL-2000. In *FBT*, pages 131–142, 2000.

[92] Tian He, Brain Blum, Yvan Pointurier, Chenyang Lu, John A. Stankovic, and Sang Son. MAC Layer Abstraction for Simulation Scalability Improvements in Large-scale Sensor Networks. In *Proc. International Conference on Networked Sensing Systems (INSS'06)*, Chicago, IL, May 2006.

[93] Lawrence G. Roberts. ALOHA packet systems with and without slots and capture. *ACM SIGCOMM Computer Communication Review*, 5(2):28–42, 1975.

[94] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. A Visualization and Analysis Tool for Wireless Simulations: iNSpect. Technical Report MCS 06-01, Colorado School of Mines, January 2006.

[95] Sam Jansen and Anthony McGregor. Simulation with real world network stacks. In *WSC '05: Proc. 37th Winter Simulation Conference*, pages 2454–2463. Winter Simulation Conference, 2005.

[96] Contributed Code - Nsnam, [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/Contributed_Code. [Accessed July 23 2007].

[97] Mineo Takai, Jay Martin, and Rajive Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 87–94, New York, NY, USA, 2001. ACM Press.

[98] Soekris Engineering, [Online]. Available: http://www.soekris.com/. [Accessed Aug. 31 2006].

[99] Xilinx Inc. *PowerPC 405 Processor Block Reference Guide*, August 2004.

[100] RF Micro Devices, [Online]. Available: http:///www.rfmd.com/. [Accessed Aug. 30 2006].

[101] Xilinx Inc. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*, June 2004.

[102] David Eckhardt and Peter Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. In *Proc. SIGCOMM'96: Conference on applications, technologies, architectures, and protocols for computer communications*, pages 243–254, 1996.

[103] tcpdump and libpcap homepage, [Online]. Available: http://www.tcpdump.org/. [Accessed Oct. 30 2006].

[104] Wireshark Network Protocol Analyzer, [Online]. Available: http://www.wireshark.org/. [Accessed Oct. 30 2006].

[105] WildPackets OmniPeek Product Homepage, [Online]. Available: http://www.omnipeek.com/. [Accessed Oct. 30 2006].

[106] Ethereal Network Protocol Analyzer, [Online]. Available: http://www.ethereal.com/. [Accessed Oct. 30 2006].

[107] Iperf: The TCP/UDP Bandwidth Measurement Tool, [Online]. Available: http://dast.nlanr.net/Projects/Iperf/. [Accessed Aug. 31 2006].

[108] Sam Bartels. WIFI Location System Investigation. COMP420Y Report, The University of Waikato, October 2005.

[109] WAND Network Research Group, [Online]. Available: http://www.wand.net.nz/. [Accessed Oct. 30 2006].