

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

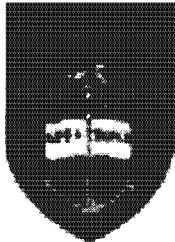
6

AAL2 Switching Node to Support Voice Services in 3rd and 4th Generation Networks

Prepared by:
Sven E. Shepstone

Supervised by:
Neco Ventura

Department of Electrical Engineering
University of Cape Town
2002



This dissertation is submitted to the University of Cape Town
in fulfilment of the academic requirements
for the Degree of Master of Science in Engineering

30 August 2002

Declaration

I declare that this thesis is my own work. Where collaboration with other people has taken place, or material generated by other researchers is included, the parties and/or material are indicated in the acknowledgements or references as appropriate.

This work is being submitted for the Master of Science Degree in Electrical Engineering at the University of Cape Town. It has not been submitted to any other university for any other degree or examination.

Signed by candidate

Sven E. Shepstone

30/08/2002

Date

Acknowledgements

I wish to thank for following individuals and organisations for their invaluable assistance:

- Neco Ventura, for agreeing to supervise this research.
- Telkom, Siemens and the Department of Trade and Industry for their generous sponsorship towards this work.
- Jon Turner, John Lockwood, Tom Chaney, John de Hart, Berkley Shands, Fred Kuhns and Ken Wong from Washington University in St. Louis, for their on-going support.
- Stuart Doyle, for providing a great deal of assistance, especially in the arena of high-performance networks and suggestions on improving the AAL2 switch design.
- Andre van Zyl, for his assistance an numerous aspects of the AAL2 technology.
- Jeanine Mc Gill, for kindly proofreading a portion of this thesis.
- My LyX document processor, for being so loyal every step of the way, and not losing a single character.
- My colleagues at the Communications Research Group at UCT, for their technical support and feedback.

Synopsis

The research community and industry alike have, over the past decade, been showing considerable interest in packet-switching networks to support voice services as well as data services. A technology that was standardised to accommodate these delay-sensitive requirements is Asynchronous Transfer Mode (ATM), which deals particularly well at transporting uncompressed voice and data. However, due to the exponential increase in wireless applications and their supporting access technologies, a need has arisen for an infrastructure in the wide area network to support and maintain the QoS requirements of low-bit rate, compressed voice. An adaptation layer known as AAL2 was re-standardised to support these specialised voice services. However, a severe side-effect of using AAL2 with traditional ATM switches results in inefficient routing and waste-age of resources.

In this study, a design for an AAL2 switching node will be proposed to address the above-mentioned issues. The design is comprised of modules that perform the following functions: Buffering, payload interrogation, protocol translations, packet classification, packet re-routing, timing, scheduling and support for signalling and management interfacing. The supporting architecture is targeted towards an embedded x86-based computing system, which itself is overlaid upon one or several ports of a high-speed, research-oriented ATM switch, known as the Washington University Gigabit Switch (WUGS).

In order to evaluate the operation and performance of the AAL2 switch architecture, a testbed is proposed and implemented, comprising the AAL2 switch at the core, with a supporting infrastructure to emulate the generation and analysis of low bit-rate voice traffic over an AAL2 connection. By conducting a set of experiments, a series of operational and performance results will be presented. Particular focus will be placed on the performance and efficiency of the AAL2 layer over ATM, as well as the ability of the switch to route packets from multiple sources to a set of output connections in the correct manner.

Contents

Declaration	i
Acknowledgements	ii
Synopsis	iii
List of Figures	ix
List of Tables	xii
Glossary	xiii
1 Introduction	1
1.1 Introduction	1
1.2 Thesis Objectives	8
1.3 Scope and Limitations	10
1.4 Thesis Outline	12
2 Literature Review and Background Theory	14
2.1 Introduction	14
2.2 Voice Characterisation	15
2.3 The AAL2 Specification and AAL2 Trunking	17
2.4 AAL2 Switching	26

2.5	Evaluation Metrics	31
2.6	ATM Switch Requirements	32
3	System Design for the AAL2 Switch	35
3.1	Introduction	35
3.2	Switching Node Functional Requirements	37
3.3	System Definition	40
3.4	System Design	44
4	WUGS Architecture	50
4.1	Introduction	50
4.2	WUGS Base Switch Overview	52
4.3	APIC NIC	53
4.4	The Smart Port Card	55
4.5	NetBSD Operating Environment	56
4.6	Integrating the WUGS Components with the AAL2 Switch	57
5	The AAL2 Switch Architecture	62
5.1	Introduction	62
5.2	System Overview	62
5.3	Operation of the SPU and Processing of Events	66
5.4	Operation of the CCU	69
5.5	Static (single-instance) and Dynamic (multi-instance) Modules	70
5.6	Functions and Organisation of the ADU	70
5.7	The PSU and Scheduling	74
5.8	Operation of the ARU	76
5.9	The Timer Unit	79
5.10	Development Notes	82

6	Evaluation of the AAL2 Switch	83
6.1	Introduction	83
6.2	Source Generator Requirements	83
6.3	Testbed Components	85
6.4	System Calibration and Time-scale Normalisation	86
6.5	End-to-end Operational Evaluation of the AAL2 CPS	87
6.5.1	Maximum Efficiency Test	90
6.5.2	Standard Efficiency Test	90
6.5.3	Low Efficiency Test	91
6.6	AAL2 Switch Evaluation	91
6.6.1	AAL2 Forwarding Test	92
6.6.2	AAL2 Path Compression Test	93
6.6.3	AAL2 Path Expansion Test	95
6.6.4	Signalling Packet Receive Test	97
6.7	AAL2 Performance Metrics	100
6.8	PDU Output Analysis	101
6.8.1	PDU Reduction with Small Packet Sizes	101
6.8.2	PDU Reduction with Standard Packet Sizes	104
6.8.3	PDU Reduction with Large CPS Packets	105
6.8.4	Summary	106
6.9	The PDU Efficiency Distribution	107
6.9.1	Distribution with Small CPS Packets	108
6.9.2	Distribution with Standard-size CPS Packets	109
6.9.3	Distribution with GSM-FR Packets	110
6.9.4	Distribution with 44-byte Packets	110
6.9.5	Distribution with 45-byte Packets	110

6.9.6	Distribution with Large CPS Packets	112
6.9.7	Summary	112
6.10	Multiple Users	113
6.10.1	Multiple Users Transmitting Small CPS Packets	114
6.10.2	Multiple Users Transmitting Standard-size CPS Packets	115
6.10.3	Multiple Users with Large CPS Packets	116
7	Conclusions	119
8	Recommendations	124
	Bibliography	128
A	Traditional Methods for Transporting Voice in ATM and SSCS Extensions	131
A.1	Introduction	131
A.2	Voice over AAL1	131
A.3	Voice over AAL5	133
A.4	SSCS Extension to AAL2	133
B	SDL Nomenclature	135
B.1	Introduction	135
C	AAL2 Switch Evaluation Test Suite	137
C.1	Introduction	137
C.2	The AAL2 Generator	137
C.2.1	Implementation of the AAL2 Generator Timeout Mechanism	141
C.2.2	Emulating Multiple Input AAL2 Paths	143
C.3	The AAL2 Analyser	144

D Abstract Data Type	145
D.1 Introduction	145
D.2 Generalised FIFO algorithm	146
D.2.1 Functions for creating and destroying FIFOs	146
D.2.2 Standard Element Processing	147
D.2.3 Miscellaneous Options (non-mandatory)	148
D.3 Generalised Linklist Algorithm	149
D.3.1 Creating and Destroying Linklists	149
D.3.2 Standard Linklist Operations	150
D.3.3 Extended Operations	151
D.4 Inverse Priority Heap	152
D.4.1 Initialisation	153
D.4.2 Standard Operation	153
E WUGS, APIC and SPC Usage Guide	154
E.1 Introduction	154
E.2 Setting up of the WUGS Controller	154
E.3 Running GBNSC and Jammer on the WUGS Controller	155
E.4 Migrating an architecture to the SPC	157
F AAL2 Switch and Client Code Overview	160
F.1 Introduction	160
F.2 AAL2 Switch Code Environment	160
F.3 The AAL2 Generator	162
F.4 The AAL2 Analyser / Receiver	164
G Accompanying CD-ROM	165

List of Figures

1.1	Using the PSTN as an Access Network	4
1.2	The Benefit of Using AAL2 for Trunking Purposes	8
2.1	Multiplexing Users onto an ATM Link using AAL1 and AAL2	19
2.2	Packing CPS Packets into 48-byte PDUs: Typical Scenarios	22
2.3	CPS Packet Nomenclature	23
2.4	CPS Transmitter State Machine	25
2.5	CPS Receiver State Machine	26
2.6	Inefficient ATM structure for AAL2	27
2.7	The AAL2 Switch	28
2.8	ATM and AAL2 Switch Synergy	29
3.1	AAL2 Duality	36
3.2	SDL Representation of AAL2 Switching	41
3.3	Switching Node System Design	46
4.1	Enhanced WUGS Port Functionality	51
4.2	WUGS Internal Organisation	52
4.3	AAL2 and ATM Switch Testbed	60
4.4	A Practical Solution for Creating AAL2 and ATM Switch Environments on the WUGS	61
5.1	Simplified AAL2 Switch Architecture	65

5.2	Passing Message Ports as Events to the SPU	68
6.1	Testbed Components to Evaluate the AAL2 Switch	84
6.2	Multiplexing Three Sources onto an AAL2 Link	87
6.3	CPS Packet Logfile showing 12 Packets, each of length 15 and CID of 17 .	88
6.4	CPS Packet Logfile showing 12 Packets, each of length 10 and CID of 20 .	89
6.5	PDU Contents for Large Timer-CU value	90
6.6	PDU Contents for Timer-CU of 20	91
6.7	PDU Contents for Timer-CU of 10	92
6.8	Analysing Data at Four Points	93
6.9	CPS Packet and AAL2 Traffic Logs from 2 Sources	94
6.10	Routing AAL2 Traffic to Different Outputs	96
6.11	Traffic Logs for 3 Sources	98
6.12	Concentrating 3 AAL2 channels on 2 AAL2 paths onto a single Output Path	99
6.13	Number of PDUs transmitted for 10-byte Packets	102
6.14	2 PDUs Required for Every 7 Packets Transmitted	103
6.15	Number of PDUs transmitted for 20-byte Packets	104
6.16	Number of PDUs transmitted for 40-byte Packets	106
6.17	Efficiency Distribution for 20 G729-10 CPS Packets	108
6.18	Efficiency Distribution for 20 G729-20 CPS Packets	109
6.19	Efficiency Distribution for 20 GSM-FR CPS Packets	110
6.20	Efficiency Distribution for 20 44-byte CPS Packets	111
6.21	Efficiency Distribution for 20 45-byte CPS Packets	111
6.22	Efficiency Distribution for 20 64-byte CPS Packets	112
6.23	Efficiency Summary Graph	113
6.24	100 users transmitting G729 10-byte Packets	114
6.25	100 users transmitting G729 20-byte Packets	115

6.26	100 users transmitting GSM-FR 36-byte Packets	117
6.27	100 users transmitting 45-byte Packets	117
6.28	100 users transmitting 64-byte Packets	117
A.1	Encapsulation of Voice onto AAL1 Payload (Adapted from ATM Forum) .	132
A.2	Encapsulation of Voice onto AAL5 Payload (Adapted from ATM Forum) .	133

University of Cape Town

List of Tables

6.1	Standard Compression Codecs	85
B.1	SDL Symbols used and their Meaning	136
C.1	Source Inter-arrival Times	142
C.2	Source Inter-arrival Times	142
E.1	Commands to establish and tear down WUGS connections	157

Glossary of Terms

For the convenience of the reader, this glossary is provided. These terms occur extensively throughout the text.

ATM Asynchronous Transfer Mode. ATM is a cell-based technology that provides multiplexing gain, can utilise high-bandwidth carrier technologies and provides service guarantees.

AAL2 ATM Adaptation Layer Type 2. AAL2 was re-standardised in 1997 for the multiplexing of short, variable length packets from various low-bit rate applications onto a single ATM connection.

AAL2 Channel A unique AAL2 channel, characterised by a Channel Identifier (CID). Several AAL2 channels are multiplexed onto a single AAL2 path.

AAL2 Path Analogous to an ATM virtual connection when carrying AAL2-type traffic. An AAL2 path may contain up to 248 AAL2 channels.

AAL2 Switching A specialised type of switching where a translation is performed based on the virtual path identifier (VPI), virtual channel identifier (VCI) and channel identifier (CID) of a packet, as opposed to the usual VPI / VCI translation that is performed in ATM switching.

APIC ATM Port Interconnect Chip. A specialised ATM chip developed by Washington University in St. Louis for sending and receiving ATM cells.

codec An algorithm capable of compressing or decompressing voice traffic. A codec may also be used to alter video streams if required.

CPS Common Part Sub-layer. The sub-layer of AAL2 that is responsible for converting between CPS-packet format and AAL2 PDU format.

FPX Field Programmable Port Extender. An FPX module contains a high gate-count FPGA on which designs may be implemented. An FPX resides between a line card and input port of the WUGS, and allows per-port packet processing to be achieved.

Packetisation Delay The length of time taken to place information into the payload of an ATM cell. A large packetisation delay can adversely affect voice quality.

Poisson A mathematical process used to model random arrivals. The probability of an arrival in a small interval of time depends only on the size of the time interval, and not on the past history of the process.

Socket A logical abstraction mechanism, where a network device is represented as a single descriptor (value). Bidirectional data flow can occur by reading and writing to this descriptor.

SPC Smart Port Card. An SPC is an embedded computer produced by Washington University for cell-based processing on a WUGS port. An SPC typically runs the NetBSD operating system and is equipped with an APIC for sending and receiving ATM cells.

Timer-CU The timer mechanism used to ensure outgoing AAL2 cells are not stalled too long at the transmitter.

VXT Virtual Circuit Translation Table. The internal WUGS table used to house ATM switch mappings.

WUGS Washington University Gigabit Switch. An 8-port ATM research switch that has been open-sourced. A WUGS facilitates the development of cell and packet-based algorithms at both the hardware and software levels.

Chapter 1

Introduction

1.1 Introduction

When communication networks began to emerge in the earlier half of the previous century, the primary form of communication that took place was in the form of speech, or what is commonly referred to as voice.¹ This was due to the fact that the technology to transmit information digitally simply did not exist at that time. However, being able to transmit voice from one location to another was soon regarded as a significant stepping stone, and soon resulted in voice-based transmissions forming an integral part of everyday life. Even in the modern-day world where other services such as video, text and data are present, being able to transmit and receive voice is still considered as the “number one” priority service by many, and hence voice based-applications have received their due share of attention. Therefore in the context of information transfer, voice has formed part and parcel of communication networks since their birth, and it is unlikely that it will become any less-entrenched in the future.

Voice, just like other services such as video and data, is generated from applications which reside at the access point to the network. This point of access need not be a physical link to the first switch in the network, but could well be, and usually is, an interface to a typical network-based Application Programming Interface (API) suited to that particular application. Applications that fall into almost any category have certain requirements (some more stringent than others) that must be met by the network technology, or else the

¹Strictly speaking, telegraphy was the first form of communication to exist.

application will perform poorly end-to-end and may even fail, and voice applications are certainly no exception to the rule. Obviously the most important requirement that must be satisfied for voice in a point to point call is somehow ensuring that the voice information reaches the other end in a certain time limit, which should never be violated. Such a time limit has in fact been suggested in the literature [29] to be no longer than 150ms. If the time it takes for voice to travel from the source to the destination, through a given network, exceeds this limit, such as in a typical telephone call for example, it may become uncomfortable and even disturbing to maintain the conversation for the parties affected at either end. Another phenomenon that can severely affect the quality of voice is known as jitter, where the voice information does not arrive in a constant form at the receiver, but instead the rate fluctuates, causing unwanted glitches. Further requirements do exist for voice applications, but it should suffice at this point to say that voice itself is quite evidently a sensitive application, and that a network technology that is deemed suitable for transferring such information should at least be able to conform to these basic requirements, namely meeting minimum delays and ensuring at the receiver that information is available and can be unpacked both on time and at a constant rate.

Traditional methods for encoding a voice stream take an analogue signal, in which the fluctuations are proportional to the amplitude of the user's speech, and digitise this information at fixed time intervals. The samples are then digitally encoded and transmitted, at a fixed rate of 64 kbps into the entry point of a synchronous network, today known commonly as the Public Switched Telephone Network, or PSTN. This method of encoding voice has been used for many decades and is in fact still in use today around the world.

In the early seventies, when the PSTN was already firmly entrenched and switching many telephone calls world-wide, parallel work had begun in North America by the Department of Defence to create a suitable redundant data network that would span the continent [1]. The protocols responsible for transferring data in this network later evolved into what is today known as the Internet Protocol (IP) and these data networks soon connected all continents on the globe to form the Internet. IP networks were, of course, originally intended for carrying data information. The intention of the IP protocol was to create a layer-3 logical inter-working technology, that would not depend on the underlying physical network architecture. These networks were vastly different to the traditional PSTN based networks in that the information (data in this case as opposed to voice) was carried in the form of asynchronous packets instead of frames filling up synchronous time-slots. Therefore these networks only transmitted data when it was necessary to do so and hence the total

bandwidth available could be allocated at all times to all the active connections, and not necessary all connections, both active and inactive. However, these networks could not operate for long before becoming congested, with the obvious result that applications' data would sometimes take longer to traverse the network than usual, and that the path taken each time through the network would not be the same. This shortcoming however, was not regarded as a problem for data as data does not have the same sensitive requirements that voice has. Two main constraints for voice that can be relaxed for data is firstly the end-to-end maximum allowed duration that will be tolerated, as well as the constant manner in which traditional voice is required to be transmitted to avoid jitter. Latency and jitter are not factors of concern for non-real-time applications, such as data. However, data transmissions cannot tolerate lost information in the network. Therefore as voice and data are radically different applications, it is scarcely surprising that two separate transmission networks have evolved over the years.

As the PSTN was around many years before data networks emerged, and since almost all homes and offices have access to the telephone, it made sense to use the telephone connection as an access technology for data networks, such as X25 networks, as shown in Figure 1.1. It was simply not possible to provide each and every customer with a direct data pipe to their home. When the Internet started becoming common-place, it was not feasible at the time to provide a separate data connection for each customer. Therefore to gain access to this central data network, customers' data was converted to an analogue form before entering the telephone network, where the digital information would eventually reach the packet-based network as data.

As the resources provided by data networks advanced, particularly bandwidth, new applications were born that tried to use these resources to provide additional services such as speech and low quality video, and hence it was no longer suitable to use the PSTN as an access network. The ISDN model was therefore soon introduced which allowed the PSTN for the first time to be a carrier for these multimedia services at a rate of 128 kbps.

In the early eighties, work had already begun on a new cell-switching technology known as Asynchronous Transfer Mode (ATM) to provide a network model that would be entirely application driven. In Europe, the driving force behind this technology was J.P. Coudreuse, from the CNET Prelude Project, who wanted to see a tight integration between services and cell-based switching networks. In the United States, Jon Turner from Washington University had been conducting extensive research on cell-switches, and their practical deployment in the enterprise [3]. Jon Turner is considered by many to be the father of

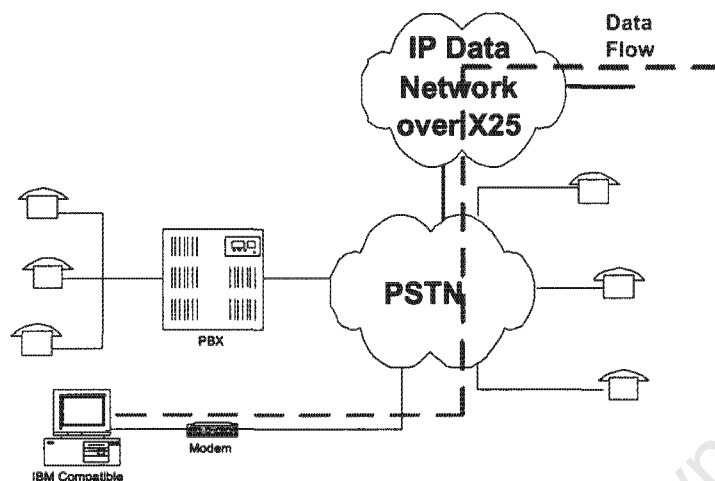


Figure 1.1: Using the PSTN as an Access Network

ATM, not because he was the inventor of this technology, but because of his driving force behind ATM research [2].

ATM, the technology of choice for the broadband ISDN model, is able to guarantee the support of four primary classes of service, which represent all possible traffic types that may occur in the network. ATM technology is independent of the underlying physical technology. Although work has been done on supporting voice calls (uncompressed voice) over ATM, the PSTN still remains the primary technology to support telephone traffic.

Before ATM technology had been seriously considered for voice, researchers had already been investigating the feasibility of transporting voice over IP networks. The interest in pursuing research in this area was sparked by the pure economies of these networks. Due to the success of the Internet, the exponential increase in the total bandwidth of these networks to support data applications resulted in decreased costs for consumers. This reached the extent where it was considered a viable option to attempt voice transmissions over data networks, as bandwidth-hungry data applications had already surpassed voice bandwidth requirements. The first requirement to transmit voice over a packet-based network was to present the voice information in a suitable form, i.e. as packets offered to the network, carrying “chunks” of digitised voice. As indicated above, IP networks are capable of offering best-effort delivery of data, and the same would apply to the treatment of voice packets, as logically, the network cannot distinguish these two traffic types in their packetised form. One way of trying to avoid congestion in IP networks, that does

not provide much guarantee, is to simply produce less data, or digital voice samples per second in this case. In order to provide voice as a lower bit-rate application than usual, compression techniques are required.

Compression algorithms usually manifest themselves as *codecs* at the originating and receiving end of the voice channel. The nature of voice itself makes it possible to implement highly effective compression algorithms, some that are capable of reducing the voice bit-rate by a factor of eight. This is due to the fact that a large portion of speech contains silence (almost fifty percent [23]) and can be completely eliminated. The silence, or instead, comfort noise, is re-inserted at the receiving end. Reducing the rate at which voice enters an IP network can help to improve the network's response in propagating the voice information, but still nothing can be said about any guarantees that the network can provide. Furthermore, if the network does suffer somewhat from variations in delay for example, buffers can be inserted at the receiving end to allow a constant play-out of the voice information. Although it is possible to transmit voice over IP, IP networks are inherently incapable of providing hard real-time guarantees that voice requires.

Up to this point we have only considered the transport of voice in fixed networks. In the striving to make applications, particularly voice, as mobile as possible, extensive focus has been placed on so-called "last mile" or access technologies, particularly in wireless devices such as cell-phones, voice-enhanced PDAs and various wearable devices. There have already been reports of the number of mobile handsets in use exceeding that of fixed devices in certain geographical areas. One limitation these devices must all deal with is very limited bandwidth imposed by scarce spectrum resources. Spectrum is an expensive resource, where only a limited band can be allocated per operator, and hence it is critical that network operators make full use of their allocated bandwidth to get good returns on their investments. Radio access technologies are steadily improving and those already successfully implemented include TDMA and CDMA. A discussion of how these operate does not fall within the scope of this document. However, it is worth mentioning that the data rates portable devices are able to sustain have steadily been increasing. Second generation or GSM portables support a data rate of 9.6kbps, GPRS supports a data rate of 168kbps (currently implementations are closer to 21kbps) and 3rd Generation (3G) devices already offer data rates of 384kbps and are expected to offer 2Mbps data rates in the near future. To maximise the limited spectrum resources that are available, one again turns to compression techniques, but in a wireless medium, compression plays a far more important role and is mandatory. A highly effective algorithm that may be deployed in present-

day 2nd Generation cellular networks is CS-ACELP (Conjugate Structure Algebraic Code Excited Linear Progression), that is capable of compressing a 64kbps voice stream into a mere 8kbps.

The drawbacks in utilising traditional synchronous networks for low bit-rate voice are obvious. There is a distinct mismatch of the information rate of the traffic source and the channel bandwidth of a network connection. Traditional multimedia services require large amount of bandwidth to operate effectively, and it is assumed the bandwidth exists to support these services. Low bit-rate services are quite different to these traditional services, and do not operate efficiently when trying to match them with existing network infrastructures. These differences help one to understand why existing network infrastructures are no longer suitable. Due to voice compression the parameters describing the traffic source entering the fixed network are vastly different to those of an uncompressed audio source. In the latter, information enters the network from a terminal at a constant bit rate (the inter-arrival time between samples is constant). Upon compression, redundant information is no longer transmitted, resulting in traffic exhibiting more variable bit rate characteristics. The burstiness of the traffic varies according to a number of factors, but is primarily affected by the codec that is utilised in the handset or traffic source. For example, as shown in [4], the GSM Full Rate (GSM FR) is a speech codec that produces a 36-bytes voice packet every 20 ms during the speech burst with a rate of approximately 13 kbps. For CS-ACELP, a 10 byte packet is produced 10 ms during the on period (during speech) at a rate of 8 kbps. As the on and off times of speech are not deterministic, the traffic produced by these codecs is hence quite bursty in nature.

The VBR (Variable Bit Rate) nature of the source implies that as traffic is only sent for a portion of the time period, the waiting synchronous network must somehow insert redundant information into information payloads to ensure that framing constraints are not violated. Another issue that needs to be taken into account is the fact that in many networks, compressed data entering the network needs to be uncompressed in order to be transmitted effectively across the synchronous network. Each time compression or decompression takes place, further unwanted delays are introduced along the connection path, which is an undesirable effect. One not only requires a network that can provide statistical gain by operating asymmetrically, but also one that can form a good core technology, where services do not need to be further mapped onto an existing underlying physical network. For example, the frames from IP traffic are often mapped onto a frame-relay or ATM transport network while traversing the public network, as there are currently no wide

area Ethernet infrastructures in place². By continually having to alter the data structure by adding additional overhead to packets as they pass to the lower levels of the network, errors, delays and protocol-mismatch become common-place.

ATM, itself a cell based networking technology, offers good support for constant bit rate traffic as well as bursty traffic. ATM has always offered support for circuit switched voice over an Adaptation Layer known as AAL1. However, AAL1 is not efficient when transmitting lower bit rate voice, mainly due to the packetisation delay. The packetisation delay, the time it takes to create an ATM cell with voice information, grows inversely proportionally to the bit rate of the voice. Therefore it will take far longer to packetise voice data from a low bit-rate source, requiring some enhancement to be made to ATM³.

For this reason, the ATM Forum has devised a new adaptation layer known as AAL2, standardised by the ITU in 1997, that caters for delay-sensitive, bursty and low-bit rate traffic, particularly voice. Essentially, what AAL2 does is to multiplex several packets (sometimes called mini-cells [23]) of low-bit rate sources into a single ATM cell before transmission. By multiplexing packets into cells, the probability of traffic being available at a given instant is much higher, with the result that cells can be sent off more regularly, eliminating delays. In the event that an AAL2 cell payload does not fill up within a specified time period, a timer will expire and ensure that the current cell, full or not, is dispatched.

While ATM is certainly the most sensible choice when it comes to multiplexing the diverse traffic types that are characteristic of today's services, it does present its own problems. Firstly the transfer delay between source to destination of a cell (or frame) is not as small as that of most circuit switched networking architectures. Secondly, ATM may prove to be less efficient than traditional TDM systems due to the cell overhead imposed [6]. It is also for these reasons that telecommunications operators who want to use ATM have been so willing to try out AAL2 technology for voice - the multiplexing gain provided by AAL2 can reduce the above side-effects of traditional ATM-based voice transport quite substantially.

AAL2 was initially intended for trunking purposes as shown in Figure 1.2. Here the voice information from various sources is packetised and multiplexed into ATM cells along a single ATM virtual connection (VC). Two problems may occur with AAL2 trunking that will be covered in more detail in Chapter 2. The first problem is the inability of AAL2 to

²Resilient Packet Ring Technologies may extend Ethernet past the local area network, but their scalability is considered limited.

³Efficiency comparisons in ATM will be covered in more detail in Chapter 2

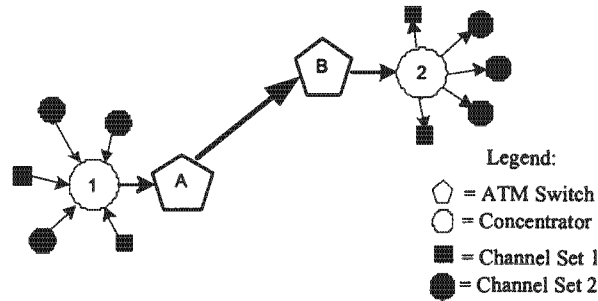


Figure 1.2: The Benefit of Using AAL2 for Trunking Purposes

multiplex voice from the output of different codecs, e.g. multiplexing voice from a GSM Half Rate codec and a CS-ACELP codec⁴. The receiver will not know what codec to apply to the retrieved packets, as it will have no idea how they were formed. Secondly, if a single ATM VC carrying separate channels is copied to two destinations, all the AAL2 sub-channels will be carried in the VC for each direction, instead of sensibly carrying only the sub-channels that are required at the destination.

These problems have resulted in increasingly more focus being placed on AAL2 switching [23]. An AAL2 switch is capable of disassembling AAL2 cell payloads into the individual short voice packets, grouping those packets destined for each separate destination onto the appropriate queue, and finally recreating AAL2 cells from the re-grouped packets before dispatching them to the following node(s) in the network. This is made possible due to the fact that each voice packet in an AAL2 connection has an additional information field, viz. the CID or channel identifier. Since the goal of AAL2 is to multiplex several channels onto a single ATM connection, a scheme must exist whereby individual channels may be identified. The CID is therefore used to assign unique identifiers to each individual channel. Therefore it is possible to perform logical AAL2 switching at the VPI / VCI / CID level instead of ATM switching at the VPI / VCI level.

1.2 Thesis Objectives

This study will focus on the design and implementation of an AAL2 switching node. It was made clear at an earlier stage that more emphasis needs to be placed on the core

⁴Suggestions have been made to use the UUI field in CPS packet headers to attempt to address this issue

functionality of the network to suit the applications that will drive the network. Alongside W-CDMA and OFDM, AAL2 has been regarded as an excellent solution to the problems and challenges imposed by future UMTS/IMT-2000 and 4th generation systems [4, 6]. There have been investigations on how far AAL2 technology should extend towards the end-user along a possible access network, but these studies will focus on AAL2 purely at the core network level, and assume existing access protocols.

Most importantly at this stage though is the question of exactly which services the proposed AAL2 switch is expected to provide. Naturally, the switch should be able to perform the AAL2 switching discussed above. It should not only perform the switching correctly (header translation) but in addition it should be robust, scalable and efficient. The AAL2 switching level should be completely isolated from the ATM level. In other words, the AAL2 network and addressing should be logical. The AAL2 switch should appear as an ATM switch to other ATM switches. The switch should also be intelligent enough to distinguish between AAL2 cells and non-AAL2 ATM cells, which shall henceforth be referred to as standard ATM cells.

As the implementation of an AAL2 switching node is highly specialised, and given the fact that AAL2 is a relatively new technology, there were no commercial tools available when the project was initiated, that would have made it possible to implement a suitable design. Although it is possible to implement a simplistic design on a high-performance FPGA, the resulting design would be non-practical, and extensive expertise would be required to provide the supporting link interfaces to allow the system to operate. Furthermore commercial vendors are often unwilling to provide customisable hardware to the research community. For this reason, Professor Jon Turner at Washington University in St. Louis had initiated a programme in 1999, funded by the National Science Foundation and DARPA, to provide a high-performance eight-port research switch to a number of universities around the world. These switches to date have been used very successfully to implement a selection of high-profile research projects (some non-disclosed)⁵. Based on the success of the programme, and the suitability of the hardware, it was hence decided to purchase one of these research switches from this university, known as the WUGS (Washington University Gigabit Switch) to implement the AAL2 switching node design. Washington University have also designed and made available an embedded computer capable of intercepting ATM cells between a link interface and the ATM switch fabric, which allows one to implement an active pro-

⁵Information on this programme can be found at the following URL:
<http://www.arl.wustl.edu/gigabitkits/>

cessing module, in this case the system architecture for an AAL2 switching node. At a later stage, Washington University released an adapter card with a programmable FPGA and supporting link circuitry, which the author and colleagues are seriously considering for future work in this field. Chapter 4 will provide a more in-depth introduction on the WUGS and associated components.

Many papers have focused on performance metrics for AAL2 technology [7, 8] and have used both analytical methods and simulation to come to conclusions how various metrics are related to the performance of a design. The intent of these studies is to compare the functionality and performance of the switching node with results suggested in the literature which will aid in obtaining conclusive results. Of particular interest will be how well the AAL2 technology for the switch scales with an increase in the number of users for a given link bandwidth, the effects on varying the packet sizes from the incoming voice streams, the efficiency of attempting to send large data packets through the switch, and what effect the AAL2 timer mechanism will play on the efficiency of the outgoing traffic payloads.

The goal of designing and implementing an AAL2 switch on the WUGS will form the first phase of a five year project which may well be continued beyond that point. The design will take into account further extensions that are to be incorporated into the switching node to enhance its functionality. The primary enhancement, which is currently being researched by a colleague of the author, will be AAL2 signalling. AAL2 signalling is a separate signalling protocol that is used in the creation and tear-down of AAL2 connections, communication of AAL2 switches in a network, updating AAL2 routes in a network and various other essential services. AAL2 signalling is a logical signalling entity and does not exist at the ATM layer. It relies on reserved AAL2 channels that are made available at system startup. A good overview of AAL2 signalling requirements can be found in [4, 9].

1.3 Scope and Limitations

While the author realises that there are a large array of factors that can enhance the performance of the AAL2 switch at hand, these studies cannot address every issue that could impact on the performance of the design.

The operating system of choice for running software on the Washington University's embedded computer is NetBSD 1.4.1. This OS was not selected by the author, but instead by Washington University as a suitable operating system on which to implement active

network designs for the Smart Port Card. Although NetBSD is a well-structured operating system and offers good performance, it is not a real-time operating system, and hence the mechanisms found in real-time systems, particularly scheduling mechanisms, which could enhance the performance of the switching node, are not possible to implement in this instance. The author has decided to use NetBSD in the implementation phase, as focus on implementing real-time mechanisms would require extensive further research and hence draw the attention away from the principle purpose of this study, i.e. of implementing a functional AAL2 switching system. However, the author is fully aware of the enhancements that a real time OS can provide, and will make recommendations at a later stage for a suitable OS.

The minimum timer resolution offered by NetBSD is only 10 ms. Although this does not affect the number of users that may be supported (a faster machine need only be supplemented to support more users), it certainly does play a role in what voice codecs may and may not be supported. To date, the codec with the shortest packet inter-arrival time is G729, which produces packets every 10 ms. However, codecs requiring better network response are continually being researched, and are likely to soon be available.

Although the WUGS is itself considered as a high performance ATM switch, it does have its limitations with regard to link speed at each port on the switch. For a very large scale implementation to support many users, the performance increase by placing the active networking modules at the input side of the switch could result in much improved performance, as the output of the module would have available of $N \times \text{Link Speed}$, where N represents the number of ports on the WUGS (currently 8 ports). Fortunately however, the combined user bandwidth in the implementation and tests is expected to be small.

Due to internal constraints and design simplification, the ATM link chip known as the ATM Port Interconnect Chip (APIC) that is responsible for transmitting and receiving ATM cells can only carry a maximum of 256 separate channels or Virtual Connections (VCs) at any given moment. This will also have a direct influence on the number of AAL2 users that can be supported. As a maximum of 248 AAL2 users are allowed per connection, this will give a theoretical total of 65000 separate AAL2 connections that can be permitted on each port of the WUGS.

1.4 Thesis Outline

The remainder of the document is organised as follows:

- Chapter 2 begins with an overview of voice, the principle application to drive AAL2 technology. A short study of the nature of voice traffic is essential, to enhance the reader's understanding of the various real-time requirements that would be placed on a transmission network. The manner in which voice traffic can be multiplexed over an ATM connection using AAL2 is then discussed in further detail, with particular reference to the transformation phase between voice packets and AAL2 PDUs. Since AAL2 promises efficient voice transfer over an ATM network, its performance against AAL1 and AAL5 is compared. The limitation of using AAL2 as a trunking technology is then investigated, and why one ought to extend a given network to support AAL2 switching. AAL2 switch concepts, as proposed by various researchers in the field, are presented, as well as performance metrics that may be used to judge a switch's operation. As the AAL2 switch will be implemented as an integral component of a larger ATM switch, ATM switch performance principles are also covered, and how they would affect the transfer of real-time voice.
- In chapter 3, the focus is shifted towards the design of an AAL2 switch architecture. The requirements in terms of supporting the transfer of real-time voice over an AAL2 connection, with an extension framework for AAL2 signalling, are first mapped into a requirements specification. This specification will show the typical path through the AAL2 switch, that would be taken by AAL2 cells and CPS packets, and corresponding actions that would be taken at each stage. A full design is presented, showing how the overall switching system may be broken up into a series of modules that operate independently from one another, and how each one is responsible for a small switch function. Finally, the interface definitions between core modules will be discussed.
- Chapter 4 will discuss the WUGS architecture and the suitability of applying this architecture to an AAL2 switch. The reader will be shown how the proposed AAL2 module can be implemented onto an embedded computer, viz. the Smart Port Card (SPC) and how the SPC will integrate with the main Washington Switch. The interface chip for sending and receiving ATM cells, viz. the APIC, is discussed in detail and several of its advantages and drawbacks are highlighted, as well as the reason why certain APIC features were made use of. Finally, the reader will be

shown how a typical test environment that is comprised of multiple components may be mapped over to a single Washington Switch.

- Chapter 5 will focus on the implementation of the AAL2 switch. Implementation choices, such as the use of threads, are justified. Each module will be presented in detail, and how the organisation and function of the module will address a set of requirements, as presented in chapter 3. Relevant issues with regard to the development of the switch will also be presented.
- In chapter 6, it will be demonstrated how the AAL2 switch will be tested. First the requirements for external testbed components will be discussed. The testbed will consist of two components, viz. a source client to generate AAL2 traffic and an analyser to receive traffic. A sequence of tests will be conducted on the switch as traffic is passed through. These tests will include establishing routes through the switch, and configuring the switch to condense and expand AAL2 routes. Finally, a set of experiments will be conducted to evaluate the efficiency of the AAL layer, and how the various performance parameters are linked to one another.
- Chapter 7 will present a set of conclusions that were drawn from these studies. These conclusions will present the order in which components were implemented, and what limitations were found to exist.
- Chapter 8 will present recommendations, which may be used to prescribe future work for an improved AAL2 switch architecture.
- Finally, a set of appendices are presented to provide further background information, as well as development specific details.

Chapter 2

Literature Review and Background Theory

2.1 Introduction

At this stage, the motivation for implementing an active switch based on AAL2 networking technology ought to be clear to the reader. In active network elements such as intelligent ATM switches and IP routers, the elements are capable of playing a much larger role in the transport and distribution of the traffic they pass around the network. Unlike passive network elements, which are merely capable of storing and forwarding data through the network, active network elements go a step further by performing some type of network transfer function on the traffic. As an example, an IPv4 router may be capable of interrogating the payload in IPv4 packets, and determining whether any IPv6 packets are being tunnelled - these may then be extracted and forwarded onto an IPv6 backbone network instead. Similarly, in the AAL2 switch being considered, the primary function of this switch is to perform AAL2 switching, as opposed to ATM switching. The only way this can be made possible is to physically remove the AAL2 sub-channels from the ATM cell stream before performing the central switching function. This can be seen as an active function, and hence the AAL2 switch can be considered as an active network element.

Before a design of any network element can be tackled, two important questions should be addressed, viz. Do all the modules comprising the switch and the switch itself function correctly, and how well does the switch perform in various operating environments, particularly those that are likely to place the system under stress. A system may be con-

sidered operational, but perform poorly under even the slightest load variations. On the other hand, a quick response time, perhaps considered as good performance, may result in unwanted alterations to the traffic and mis-insertion (switching of data to the incorrect output ports).

If the AAL2 switch is ever to perform adequately in any networked environment, it is necessary to have a good understanding of the type of application that will reside at the end points of the AAL2 switching network, in this case, primarily low-bit rate voice. This is of particular importance to ensure that the required mechanisms can be made to conform to a certain specification. One cannot set standards for network elements where the traffic is only vaguely understood. Hence a more detailed overview of the characteristics of voice traffic will be covered to aid the reader in understanding why components must meet minimum application requirements.

In order to address the issue of functional correctness, a thorough understanding of the technologies upon which the switch will be based is mandatory. A significant portion of this chapter will therefore be devoted to the AAL2 technology itself, and how AAL2 can be realised initially as a trunking technology, and later in a switching technology. Research in the academic community has significantly aided cellular operators in making decisions whether to try AAL2 technology for their core networks, due to its suitability in supporting voice applications [23]. It is therefore imperative that one takes a look at such research in this chapter, in order to determine the current status of AAL2 technology, and to show what areas in this technology still requires attention. Various papers in the literature have suggested suitable metrics on which to base the performance of a typical AAL2 switch, and it will be shown how the results of these will assist in the evaluation of the switch architecture at a later stage.

2.2 Voice Characterisation

The previous chapter introduced voice as a sensitive real-time application. Voice is particularly sensitive to end-to-end delays imposed by the network as well as delay variation. Some have argued that mean delay for voice is not a meaningful QoS measure for end-to-end voice transport [19]. Instead, it has been suggested that one should rather examine the tail of the voice delay distribution, and quantify the overall fixed delay at the K^{th} percentile. When one wishes to analyse the side-effects suffered by voice when passing through

a network, what is often more of interest is the ratio of successful, or uncorrupted packets that reach the destination over the total number of packets transmitted. As delay is the main contributor to voice degradation, we are therefore concerned with what percentage of packets will suffer as a result of a given fixed delay in the network. Clearly, increasing the delay will result in more packets arriving late at the destination, which may be unusable by the application. If one wishes to maintain a certain success rate, it is possible to vary the delay in the network, until we find the success ratio we require.

To reduce jitter on voice networks, buffers can be inserted at the receiving end. The purpose of such a buffer would be to ensure clock-recovery at the receiving end. If the buffer is too small, the receiver will be unable to smooth out the traffic successfully. If the buffer is too large, cell delays are once again introduced. The rate at which information is played out at the receiver will depend on the voice codec that has been selected to decode the information.

A subjective metric has been developed to evaluate voice quality, known as Mean Opinion Score (MOS). Various subjects are requested to evaluate the quality of voice in a session, and give a score from 1 to 5, where 1 is considered unacceptable quality, while a score of 5 is given to excellent voice quality. A lower score naturally indicates that more effort is required by the subject to interpret what is being said. A MOS of 4 is regarded as toll-quality voice. The MOS scale is often used to indicate what quality is sustained in the network.

When a voice source is compressed, two additional requirements are placed on the network. Firstly the rate at which the voice enters the network now varies, as opposed to the previous constant bit-rate stream for uncompressed voice. We refer to such voice traffic as variable bit rate traffic or VBR traffic, as opposed to constant bit rate traffic, or CBR traffic. The reason for this VBR nature is simple: compression algorithms are designed to remove redundancies in voice. Therefore during a period of silence there is no need to transmit any information, and hence the traffic source will remain idle during the period [18]. This would imply that, if one were to look at the shape of the traffic source over time, it would look very irregular, i.e. the envelope would contain many peaks and troughs, and would not look even. When the bandwidth level of a source is at its lowest (corresponding to a trough), it would be consuming the least possible bandwidth of the link, and hence more bandwidth would be available for other sources. Therefore during silence intervals, the network can make use of multiplexing gain to transmit the voice information from another source. However, when a user suddenly speaks, the network will be required to respond

much faster and have a large instantaneous bandwidth to handle this new talk spurt.

The second requirement placed on the network when compressing voice is that the network should be more intolerant of any loss of information. In uncompressed voice and video, the loss of a few bits of information will have a minimum effect on the perceived quality at the receiving end. However, if a compression algorithm compresses a voice stream by a magnitude of four, losing one bit of information will be significant to losing four times the amount than in the uncompressed case, and the results in loss of quality will be four times that at the receiver. Therefore, the higher the compression ratio being achieved, the higher the network should prioritise the voice traffic path.

In the literature, numerous comparisons have been performed with telephony over both best-effort IP and ATM networks [30]. The large number of hops a packet can experience in an IP network will affect the end-to-end delay. Furthermore, the non-deterministic path that packets follow through the network will contribute to the overall delay variation (jitter), hence making it impossible to guarantee the quality of service (QoS) required by voice applications. The best performance achieved with voice over IP occurs in an under-utilised IP network, where compression is performed to lower the bit rate. For small intranets, this may appear to be an acceptable solution (simply provide more bandwidth) but in a core networking environment this is clearly unacceptable.

2.3 The AAL2 Specification and AAL2 Trunking

AAL2 was initially named AAL-CU (Composite Users) but the original technology that had been suggested for the segmentation and re-assembly of voice had become unpopular over the years, particularly due to its inefficient handling of voice packets. The ATM Adaptation Layer Type 2, or AAL2 was initially proposed by the ATM Forum and soon standardised by the ITU-T in late 1997, to support the delivery of short packets from variable, low bit-rate and delay-sensitive applications[15]. This newer technology has completely replaced AAL-CU, and over the last number of years, has been drawing widespread interest from the research and industrial community.

The traditional way to encapsulate voice into ATM cells makes use of AAL1. AAL1 was suggested as a suitable adaptation layer for ATM for supporting circuit switched voice connections. As frames or packets of voice are sequentially removed from the input source, the bits are placed into the 48 byte payload of the ATM cell. Once full, the ATM cell

is dispatched. In this case, the cell only holds data from one source. For 64kbps voice, where samples are arriving relatively quickly, the time taken to fill the ATM cell would be relatively short, and hence the *packetisation delay* will be small. However, packetising information from lower bit-rate, compressed voice sources has a serious effect in increasing delay [21, 19, 6, 14]. This is due to the fact that the traffic is bursty, and hence periods of time exist during which no information is generated. Furthermore, on average, voice information is arriving at a much slower rate than before, and hence the time taken to fill the ATM cell is extended. As an example, the time taken to fill an ATM cell from a 64kbps voice source using AAL1 is approximately 6ms [6]. However, when reduced to 8kbps, the packetisation delay is increased to 48 ms, eight times as much. The packetisation delay is therefore inversely proportional to the bit rate of the voice source being considered, thus rendering AAL1 quite inefficient for low-bit rate voice. It is quite interesting to note that ATM using AAL1 for voice is actually less efficient than traditional circuit switched networks, due to the cell-header overhead [16].

AAL2 has been designed to alleviate this drawback by multiplexing several low bit-rate connections into the payload of a single ATM cell. The chance that three voice sources are idle at any given moment is far lower than that of a single source. This implies that the time taken to fill an ATM cell from more than one source will be significantly shorter. As one is multiplexing several voice channels on a single ATM VC, further information is required to distinguish these channels on an end-to-end basis from one another. For this reason, each channel is identified by a separate ID, known as the Channel Identifier (CID), that is stamped onto the header of each packet carrying traffic for this channel. The principle of carrying multiple voice channels over a single VC as opposed to the first generation scheme of one VC allocated per channel, is shown diagrammatically in Figure 2.1 below.

Although AAL1 and AAL5 are not primarily intended for transmitting low bit rate voice over an ATM connection, it is still interesting to compare the efficiencies of these adaptation layers to that of the AAL2 layer. Briefly, AAL1 is intended to support low-delay CBR applications that are typically synchronous in nature. If the source information is structured, it is capable of restoring this structure at the receiver. In addition to this, AAL1 includes clock recovery mechanisms at the receiver to ensure synchronous operation [1]. AAL5, on the other hand, is intended for the transport of VBR data frames, which are typically longer than the ATM cell payload size of 48 bytes. These frames may be

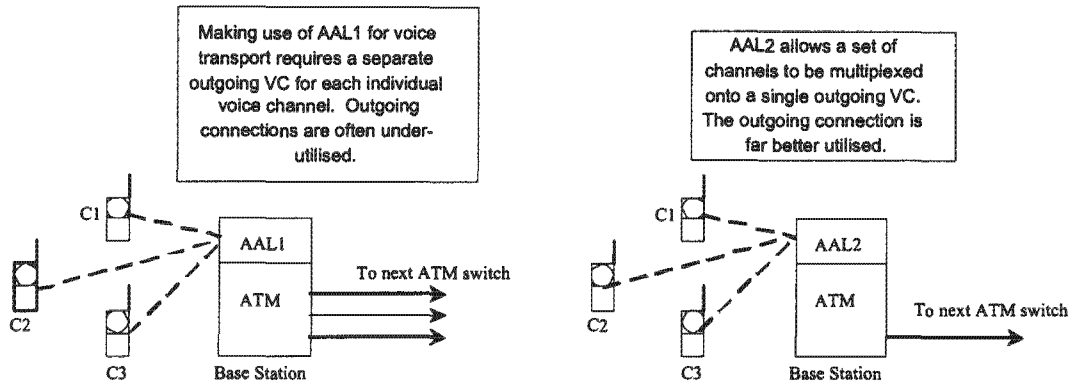


Figure 2.1: Multiplexing Users onto an ATM Link using AAL1 and AAL2

transmitted in either a streaming mode or message mode, depending on the application requirements. The reader should be aware that neither AAL1 nor AAL5 offer multiplexing mechanisms in the Common Part Sub-layer (CPS), as there is no way of uniquely identifying separate flows on the same virtual connection¹.

Various simulation models have been proposed to analyse the efficiency of AAL1, AAL5 and finally AAL2 voice sources [19]. In this study, this author assumes homogeneous voice sources, of varying bit rates, which are then applied to the input of his model. In the paper, this author is interested in the number of voice sources that may be supported, without violating a specified quality of service, which is given as a bounded 95 % percentile of voice, as described above. These experiments are repeated for all three adaptation layers, by varying either the bit rate or voice packet size. The results from these experiments are interesting, but not surprising. [19] has found that for uncompressed voice of 64kbps, AAL1, AAL5 and AAL2 are equally efficient, but for low-bit rate voice of 8kbps, which can be obtained using CS-ACELP, AAL2 is approximately five times more efficient than both AAL1 or AAL5, and that for voice packets up to 40 bytes in size, AAL1 and AAL5 perform identically. Furthermore, this paper concludes that for low bit rates, the total number of users than may be supported is very sensitive to the packet size chosen, while for high voice coding rates, varying the packet size does not necessarily increase the number of users that can be supported.

Other papers have investigated the possibility of increasing the number of users on an AAL2 channel by implementing several techniques that may reduce the quality of the

¹Note that LLC may provide multiplexing facilities, but at a higher-layer than ATM.

voice somewhat. Analytical performance models and results for AAL2 multiplexors that implement bit dropping on certain voice packets have been presented [15]. The author has assumed 32 kbps voice that would be produced by a ADPCM codec as defined in ITU-T G-727. Bit dropping is possible with AAL2 firstly due to the fact that one has access to the individual voice packets in the channel and secondly, for each channel, the information is arranged with the least significant bits containing the information of least relevance to the quality of the traffic. Dropping the relevant bits in the packet is far superior to previous techniques of employing cell dropping, which introduces once again unwanted packetisation delays [4]. Furthermore it is not easy to work out how many packets are being dropped in a cell, or the significance of the information loss. The bit-dropping model of above has suggested significant higher capacity, e.g. for $\frac{1}{4}$ T1 channel. Applying these techniques almost doubles the number of users that can be supported, and approximately 30 % more users can be supported on a full T1 connection [15]. It was shown that it is possible to implement the bit-dropping shown above, and yet still retain a MOS of 4.

Still others have examined the suitability of AAL2 for transferring data packets. When 3G networks utilising AAL2 technology in the core have acquired a critical mass of consumers, value added services (text, low bit-rate compressed video, file-transfer) are likely to come next. It may be more feasible to transport these data services over AAL2 to simplify channel provisioning and billing in the AAL2 network. Furthermore, the need for dynamically creating AAL2 connections has given impetus to AAL2 signalling, which itself can be regarded as AAL2 data². For these reasons, simulation models to investigate the possibility of transporting data over AAL2 have also been presented [4]. They have found that although AAL5 has slightly less overhead for packets shorter than 500 bytes, the difference in percent overhead between AAL2 and AAL5 for packets larger than 750 bytes is constant. Therefore, provided a Service Specific layer exists above the Common Part Sub-layer (CPS) for AAL2, it would be possible to segment these large packets for AAL2, and yet not sacrifice too much efficiency.

These findings will be valuable to the author in evaluating the effectiveness of the AAL2 technology considered for the switch. One is particularly interested in how the switch will respond to the range of traffic types suggested in the above simulations, and for each traffic type or codec, the number of users that may be sustained. Although these authors have suggested suitable efficiency metrics, they have focused on simulation models and analytical

²Some signalling messages can be 129 bytes long, which is more than twice the maximum payload length (64 bytes) that is supported by AAL2

derivations. Traffic sources have either been Poisson [8] or Markov modelled [16], the talk period to total period ratio has been explicitly stated, processing delays in constructing AAL2 cells have been ignored, packets lengths and number of cells per packet are assumed constant [8] and in some cases queues for ATM cells are considered infinite. It is therefore of interest how real-world traffic and components will cooperate, and whether the efficiencies, delays and sustained load of a physical AAL2 switch will mimic these results or not. In a typical implementation, queues are expensive (memory may not always be available) and each queue adds unwanted delays. In addition, processing delays are very real in an AAL2 implementation, and if the wrong approach is opted for, this will impact no doubt on the performance.

It would be intuitive at this stage to examine briefly the nomenclature and structure of AAL2 as it currently stands. Further details may be found in the ITU recommendation ITU-363.2, on which the following information is based. This adaptation layers is primarily responsible for segmenting information from upper-layer applications, multiplexing this information if necessary, and passing the information in a suitable form to the ATM layer for transmission. AAL2, as in other adaptation layers contains the Common Part Sub-layer (CPS) and Service Specific Convergence Sub-layer (SSCS). The SSCS defines additional services that may reside on the top of AAL2, e.g. frame-based services such as Internet traffic, and mechanisms for additional error detection and correction. In many cases, particularly for short voice packets, the SSCS may be null, and as it is not required in the interim, a detailed functional overview will not be given. The CPS contains the mechanisms to segment packets that are generated from either a voice or data source. Provided these packets do not exceed 64 bytes they may be packetised into a sequence of one to three ATM cells, depending on the packet length, and where in the ATM cell payload the packet was started. Figure 2.2 shows examples of packets allocated to various cells.

The CPS packet is defined as the packet that will hold information for a particular source. A CPS packet has a three-byte header and variable size payload, as shown in Figure 2.3. The three byte header has 24 bits split into four fields, an 8-bit field, a 6-bit field, a 5-bit field and another 5-bit field, representing the Channel Identifier (CID), Length Indicator (LI), User-to-user Indication (UUI), and Header Error Control (HEC) respectively. As AAL2 is required to multiplex several of these packets into a single cell, packets for each ATM VC are uniquely defined with the CID. However, although a theoretical limit of 256 connection could be supported, some of these are not used or reserved, and hence only channels 8 to 255 may be supported [12]. The CPS packets have variable lengths and

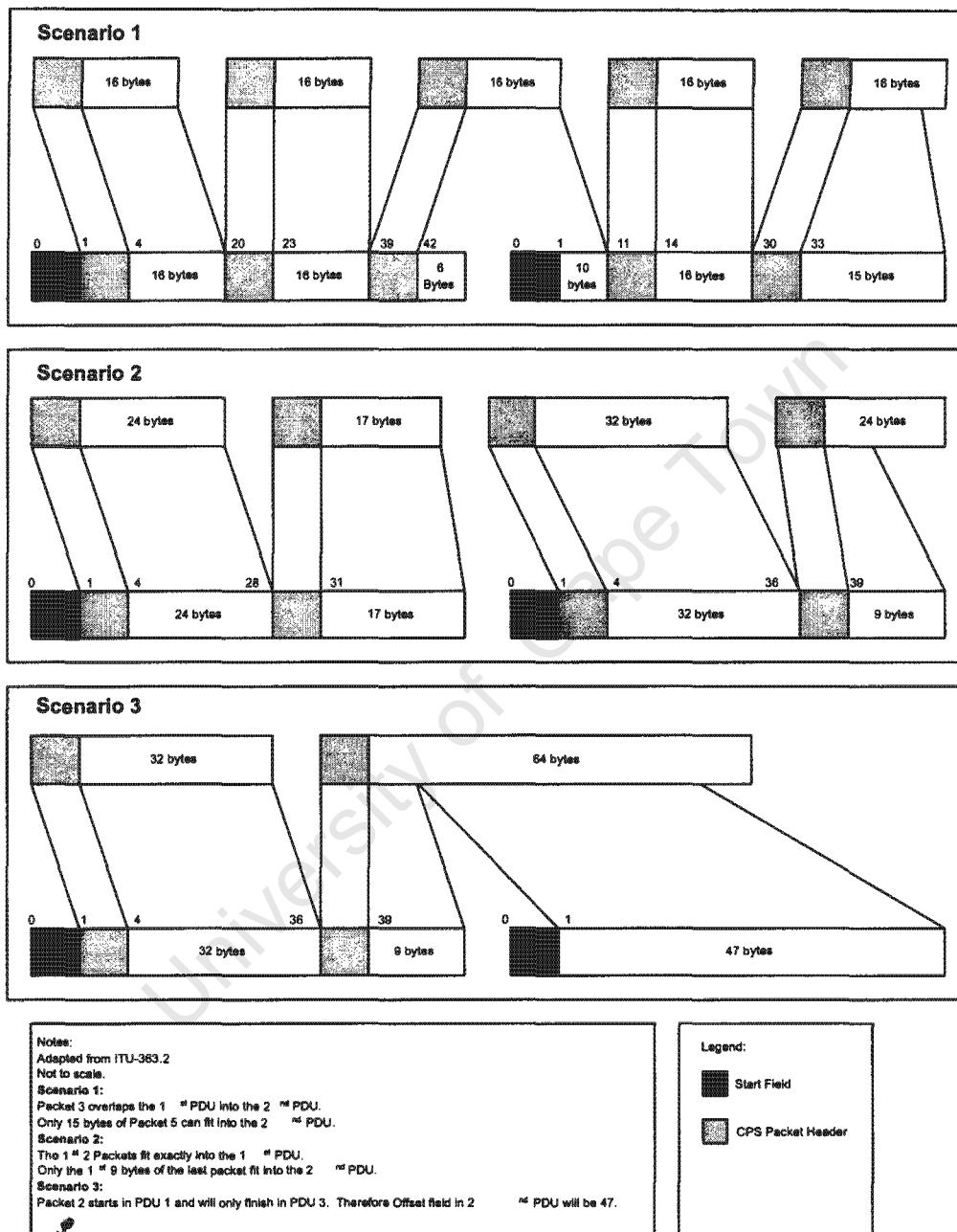


Figure 2.2: Packing CPS Packets into 48-byte PDUs: Typical Scenarios

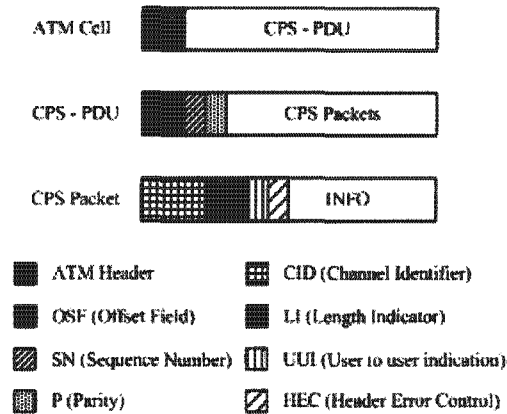


Figure 2.3: CPS Packet Nomenclature

hence the LI field can specify a length of 1 octet to 64 octets. The LI field is always one less than the packet payload size. The UII field is passed transparently through the ATM stack by the AAL2 CPS to the corresponding CPS layer on the receiving end, where it is then passed up to the SSCS (if it exists) or the application. Hence applications may use the UII field as they see fit as the structure and contents of this field is not specified in this recommendation, e.g. the UII may be used to convey what codec ought to be applied to the receiver to uncompress the voice packets, as pointed out earlier. Finally the HEC field is used to detect any possible errors in the packet payload³.

The AAL2 PDU (Protocol Data Unit) is a 47 byte structure that is capable of holding zero or more of these CPS Packets. Therefore, in the ATM cell payload, there is one byte left over for what is known as the start field. The start field contains an Offset Field (OSF) which is 6 bits, Sequence Number (SN) and Parity (P) which are each one bit long. The OSF indicates where a new packet will begin in the CPS PDU. This is due to the fact that the bytes immediately following the start field may be the continuation of a packet that started in the previous PDU. The header of this packet with its LI field would have been in the previous PDU as well, and hence no length information would be available in the current cell, which is why one requires this offset field. If a new packet starts directly after the SF, the OSF will be zero. When there is no more information in the CPS PDU, a padding of zeros is placed in the remaining octets. The SN is an alternating bit that allows

³There is insufficient information in the HEC to form any correction on this data.

the receiver to determine whether an ATM cell was dropped in the network. A shortcoming of this scheme is that an even number of cell losses is undetectable [23]. The parity bit is an odd parity bit calculated over the start field and allows the receiver to detect errors. The start field, together with the AAL2 PDU, itself containing various CPS packets, are concatenated to form the ATM PDU. Finally the ATM cell header is appended to the front of the ATM PDU before being transmitted through the ATM network.

If the SSCS exists above the CPS, the interface between the two will contain a service access point (SAP) for each different service requested from the above layer. The above layer will then pass its information (CPS Packets) as a service data unit (SDU). It is the task of the CPS layer to multiplex several of these SDUs into AAL2 PDUs to be passed eventually to the ATM layer. For this, the implementation of a state-machine based CPS transmitter is recommended [15]. On the receiving end, a mechanism is required to perform the reverse process of extracting individual packets out of the cell stream, and for this purpose a similar state-driven CPS receiver has been recommended. The exact operation of both the transmitter and receiver would fall completely out of the scope of this document, and hence the reader is urged to consult the recommendation where an SDU-based design (System Definition Language Design) can be found. However, a brief overview of each will be given.

The CPS transmitter contains four states, Idle, Part, Full and Send, as shown in Figure 2.4. If the transmitter has not begun with the process of constructing a 48-byte ATM PDU, it will remain in the Idle state until the transmission of a CPS packet is requested. When this does eventually happen, a special timer is started and the transmitter begins the process of constructing the new PDU. This timer is known as the timer-CU. When the transmitter has finished processing this packet it will jump to state Part, until either the timer expires, or the AAL2 PDU is filled. At an earlier stage this document stressed the damaging effects on voice quality caused by network delays, of which packetisation delay in low bit-rate voice is usually the culprit. Therefore, once the creation of a PDU is commenced it would be non-sensible to wait too long for an additional voice packet to arrive, as this would violate the time constraints of the first packet, which would have already been mapped to the first series of octets in the new cell. It is for this reason that the use of the timer-CU is recommended. When the timer-CU expires, it indicates to the system that it should not wait for any further information, and that the half-empty PDU should be immediately dispatched. In such a case the PDU should be filled with padding information, as indicated above. When the value in the timer-CU reaches its limit, the state

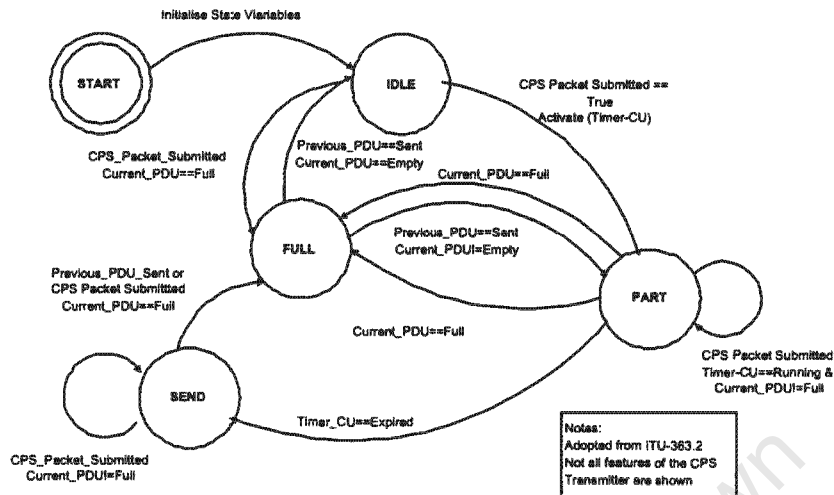


Figure 2.4: CPS Transmitter State Machine

machine is immediately switched to state Send, where it will await permission from layer management before sending the completed CPS-PDU. If the CPS-PDU is filled before the timer expires, the transmitter will once again await layer management's request to dispatch the cell.

The CPS receiver resides at the receiving end and contains only one state, Idle, which is shown in the state diagram of Figure 2.5. After processing each AAL2 PDU and extracting the packets, the receiver will return to the beginning of the Idle state to await the following PDU. Upon receiving a PDU, the parity and sequence are checked and if the PDU contains any outstanding data of a CPS packet already being formed, this information is extracted and appended to the packet. Otherwise, any further information is extracted to form new packets. If enough octets are extracted to match the LI field in a packet header, the packet is considered complete and dispatched to the upper-layer application or SSCS. If the end of the PDU is reached before enough octets can be extracted, the remaining data can be expected in the following CPS PDU. Note that no timer is specified or needed at the receiver - its operation is purely asynchronous.

Such a state machine has in fact already been used in simulation models [19]. A transmitter and receiver on the Linux operating system has already been successfully implemented, where ATM drivers were modified to support AAL2 [17]. Others have examined the timer-CU more closely and found that the number of bytes wasted as a result of the timer-CU is inversely proportional to the number of users on a connection [23]. Hence the more users,

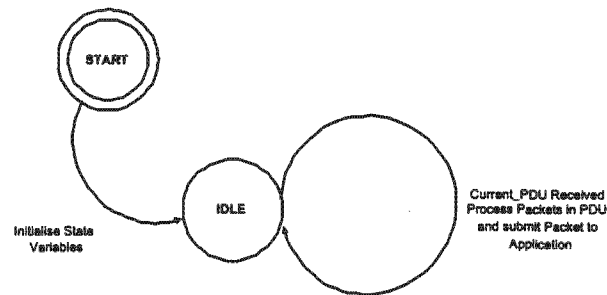


Figure 2.5: CPS Receiver State Machine

the less critical is the operation of the timer-CU to the system. It was shown that for under 50 users, the timer-CU was critical to preserve the voice quality of each channel. In Chapter 6, we shall see that the stage where the timer is not required is around the 47-users level, which is almost identical to the result above. Typical values suggested in the literature for the timer-CU range from under 2ms to 7ms.

The efficiency and simplicity of AAL2 for voice has given impetus to the design of many AAL2 trunking systems, where the channels of various AAL2 users are concentrated at one end of the ATM network, and extracted at the receiving end. By AAL2 users, the author is referring to applications that would generate low bit-rate voice. Thus an AAL2 user could be the application residing on a mobile handset about to carry the data over wireless ATM, but more typically is voice data in AAL2 form leaving a base station over a fixed ATM link. Many organisations have investigated the possibility of PABX systems utilising AAL2 technology, where the AAL2 users can be regarded as telephone handsets. For organisations already employing IP telephony who wish to switch over to AAL2 technology, no immediate modification in their telephone handsets would be required, but instead mainly in the voice concentrators. This is due to the fact that provided a suitable SSCS exists above the CPS in the AAL2 enabled PABX, extended frames (IP frames) can be passed successfully through the ATM network as AAL2 cells. Organisations that now actively market AAL2 technology include RAD, Netopia, Siemens, Ericsson and Cisco.

2.4 AAL2 Switching

For small networks and channel pipes that simply carry traffic from one end to another, AAL2 is currently the most efficient technology available. However, as the network size

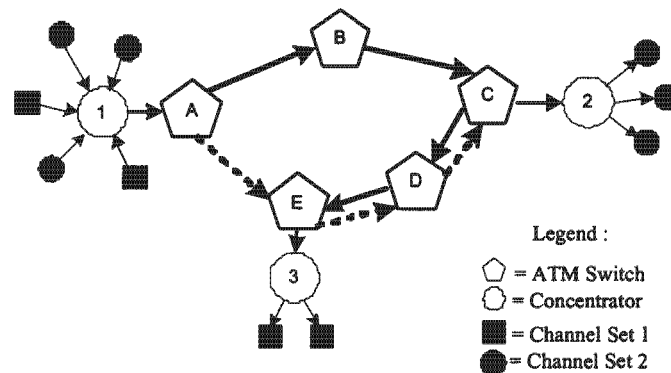


Figure 2.6: Inefficient ATM structure for AAL2

increases, AAL2 does not always scale well, and can start to cause severe headaches for network planners. By network size, one does not necessarily imply just an increase in the number of users. Simply increasing the number of users on an end-to-end channel may allow AAL2 to perform more efficiently, in terms of increased multiplexing gain achieved, as shown previously. However, if the network grows geographically larger, eventually more and more endpoints are required. Such a network setup is shown in Figure 2.6.

This diagram shows a configuration of five ATM switches, three AAL2 concentrators and various voice sources/receivers. The first AAL2 concentrator receives five different voice channels for transmission. The channels have been arbitrarily divided into two groups, to emphasise that there are in fact two different destinations for the various voice channels. Three channels are to end up at concentrator 2 and the remaining two channels at concentrator 3, where they will then be de-multiplexed.

The initial problem experienced by simply utilising AAL2 as a trunking technology is that a single VC (Virtual Channel) is only intended to carry traffic that has the same pair of cell-assembly and cell-disassembly devices at the source and destination. This means that traffic from different cell-assembly or cell-disassembly devices cannot be mixed into a single VC. If five voice sources enter an AAL2 network, each having being coded in a separate way, it is vital that the AAL2 packets are sent to the receiver with the correct codec information to decode or decompress the packets correctly, otherwise the recovered traffic information will be meaningless.

Another phenomenon that may be experienced with AAL2 is depicted in the following

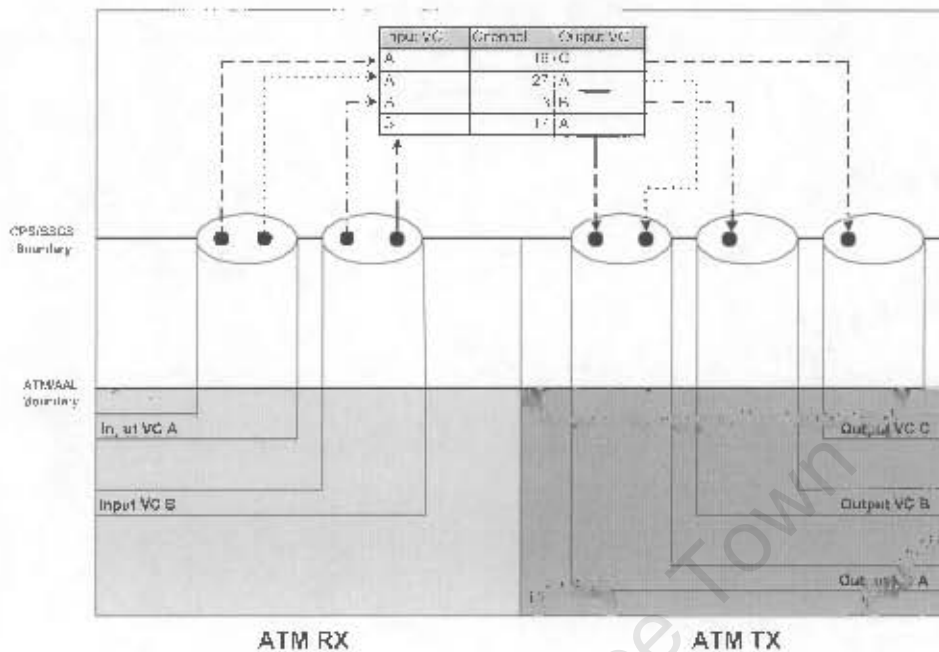


Figure 2.7: The AAL2 Switch

scenario. Looking back at Figure 2.6, we are presented with two choices at ATM switch A: Either switch the five voice channels along the solid line from switch A to switches B, C, D and finally to switch E, thus undergoing four ATM switch hops. Alternatively, traversing the dotted line results in a total of three hop counts. Regardless of what route is taken, each concentrator receives five voice channels and ends up discarding at least two. The network example shown in Figure 2.6 is more practical and likely to appear in a typical ATM network than the idealised trunking example of Figure 2.1, where each and every voice packet carried on the entire route is intended for the same destination.

An ATM switch switches at the ATM layer by modifying the ATM header and is unaware of the AAL2 packet structure within each cell. As indicated in the previous chapter, an AAL2 switch is capable of disassembling the incoming ATM cells into individual CPS packets, and switching based on the CID found in this header of each packet, as well as the VPI and VCI values in the ATM header. Cells leaving the AAL2 switch contain packets all destined for the same destination⁴. This functionality is depicted in Figure 2.7.

A typical network topology incorporating both ATM and AAL2 switches at the periphery

⁴This is provided further downstream AAL2 switching is not required.

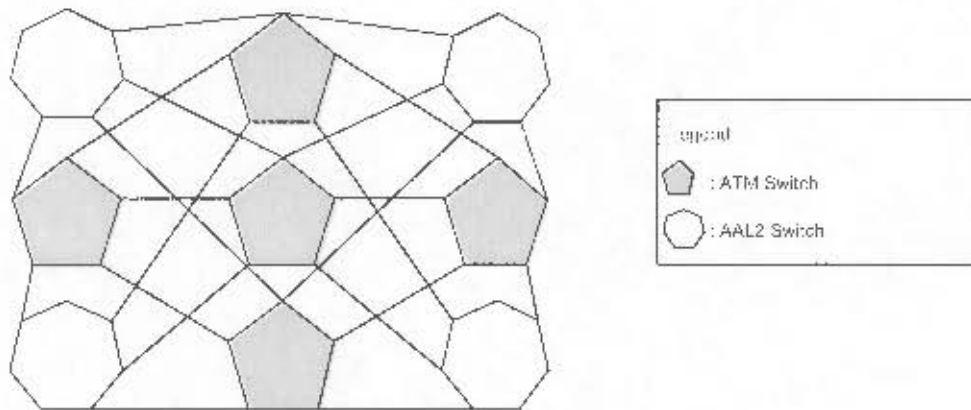


Figure 2.8: ATM and AAL2 Switch Synergy

of the network is given in Figure 2.8. The earlier along the network path AAL2 packets can be broken up and reorganised, the sooner these packets will traverse their destination along a more optimised path, resulting in both reduced network delays and less load on the network in carrying unwanted voice traffic.

The design of a monolithic, VLSI (Very Large Scale Integration) single-chip AAL2 switch was proposed in 2000 [14]. The author of this paper compared two types of switch, Type I and Type II. A Type I switch is able to break up a CPS Packet into two parts whereas a Type II switch cannot. The paper shows that the synthesisable logic for a Type I switch requires substantially more on-chip resources as Type I entails a more complex re-assembly process and may introduce delays in the Content Addressable Memory (CAM) used for buffers. The location for Type 1 AAL2 switches is suggested to be at endpoints of the network and Type 2 switches should reside in the core [14]. Furthermore this author uses simulation techniques to compare metrics for both types of switch, including time taken to discard cells, suitable CPS packet buffer size to maintain a certain ATM cell rate and the typical loss ratio experienced with CPS packets when timeouts occur. In these studies, voice entering the switch is assumed full-rate (64 kbps), which does not really take advantage of the efficiency offered by AAL2.

Others have focused more on performance issues of an AAL2 switching network and have derived results from analytical methods [21]. In their analysis, a fixed CPS packet length is assumed and arrival-rates for packets are Poisson modelled. State space theory is used to derive the cell delay variation (CDV) and analytical expressions for both the delay distribution as well as the PLR (Packet Loss Ratio) in the AAL2 switch. For a single node

the paper investigates performance issues such as load, bandwidth, timers and buffers. The timeout value in AAL2 switches is critical to delay performance and buffer size is critical to loss performance [8]. Finally, a network model incorporating AAL2 switches is proposed, where the reduction in bandwidth that can be achieved is shown by placing AAL2 switching nodes at strategic points.

A large scale VTOA (Voice and Telephony over ATM) switching node has also been proposed [18]. Here instead of identifying the internal mechanisms required in a switching node, the authors have instead investigated the feasibility of migrating VTOA nodes with existing network technology, particularly that found in the public switching network. The studies present three approaches for implementing a large scale VTOA architecture. The first is an STM based switching node, where a VTOA handling node is inserted for each outgoing ATM trunk. The ATM based switching node allows incoming calls to be mapped to virtual connections in the public ATM network, and may either incorporate STM switches (which concentrate incoming calls over SDH channels) or AAL2 switches (which first compress the incoming calls into ATM cells, one call per cell, before multiplexing many voice streams into single outgoing cells). For these three implementations Common Channel SS7 (CCSS7) signalling has been suggested to reroute calls to the STM network in the event of a fully-loaded ATM based VTOA node refusing a call connection. Services that should be offered by VTOA-capable technology for both the consumer and network operator have also been defined [18], and the economic implications of implementing VTOA with an existing network has been analysed. Customers are unlikely to accept the newer technology if the cost per call is not less than that of a standard 64 kbps call or if the quality of voice calls is degraded unacceptably while passing through the network.

It is thus evident that sufficient research has been done on AAL2 technologies and AAL2 switching to determine what performance metrics should be included for evaluation, and that one should take these metrics into account when designing these various components. Apart from the VLSI architecture that was implemented on a single chip, most authors have either made use of simulation techniques or analytical methods to produce results, and the author is not aware of any research-based attempts to produce a fully-fledged AAL2 switching node. In the VLSI approach, the authors did not mention how their chip-based design could be incorporated into a standard network, and what supporting components would be required. Furthermore, for analytical derivations, assumptions are often required to allow researchers to formulate a mathematical model, as shown above. Unfortunately many of these assumptions do not always mimic true characteristics and requirements of

typical applications, and hence it is necessary to evaluate a physical implementation of such a switching node and its supporting AAL2 components. None of the papers thus far have focused on how one maps these system components into a possible software-based approach, such as an embedded computer and hence possible side effects that may result from the use of a non-real-time operating system used for the implementation. The designs presented above also do not consider the addition of value-added services, or more importantly, signalling upgrades. AAL2 signalling has come under the spotlight recently, and co-research is aimed at producing an AAL2 signalling stack to complement the AAL2 node functionality.

The results obtained from simulation and analytical methods are however extremely useful for comparison with the real switch, and it is the goal of the author to produce such a switch, based on a real ATM switch, and determine its behaviour under many of the conditions imposed on the simulation models developed above. Furthermore by using a ATM switch already developed by others, focus can be placed on the AAL2 functionality and not the ATM functionality, which ought to both be isolated from one another in any event.

2.5 Evaluation Metrics

Results obtained from evaluating the switch's performance under various conditions will be compared to performance metrics obtained from the literature as indicated above. The author is interested in investigating the following aspects, some of which are functional and others performance related:

- Large numbers of AAL2 users: For a given link bandwidth, the effects of introducing large numbers of users that can be supported by the suggested AAL2 switch will be investigated. For each new user, a separate CID is required. There is a limit to the number of CIDs (248) per connection. Therefore the number of connections that may be supported on the ATM switch would affect the number of users in an industrial environment. It has also been suggested that the number of users that may be sustained can be very sensitive to the maximum allowed packet size.
- Multiplexing gain: The multiplexing gain for AAL2 increases with the number of users in the network, as well as maximum allowed packet length, traffic bit rates and specified timeout values.

- Efficiency: The efficiency of the payload of outgoing AAL2 PDUs is dependent on a number of factors, including packet-size, timer, and the number of users on the connection.
- Timer-CU: In the literature various values for the timer-CU have been suggested. It was also stated that for 50 users or less, the timer-CU's presence is critical. The operation of this timer-CU in producing minimal network delays, but still maintaining a suitable multiplexing gain will be investigated
- Finally, the application of AAL2 to support large data frames such as those found in IP traffic will be analysed.

2.6 ATM Switch Requirements

When cells enter the AAL2 switch, AAL2 processing is performed at the input of the switch. Once cells have been processed, they enter the ATM switch fabric where the cells headers are translated to the appropriate output connection. As there is an AAL2 switching component and an ATM switching component, we cannot ignore the ATM switch sub-components that will no doubt have a large impact on the performance of the AAL2 switch. As an example, if sufficient buffering is not provided in the ATM switch, bursts of input traffic will result in the AAL2 switching module not being able to process this traffic fast enough (it is held up by the buffer-less switch), and hence information will be lost. We therefore conclude this chapter by giving a brief overview of the fundamental ATM switch components, and what effect these components would have on an AAL2 switching system:

- Buffering - Buffering forms an essential element in most ATM switches, particularly due to the fact that it is expensive and non-trivial to implement a non-blocking switch architecture (which in theory, would not require any buffering). As the traffic most likely to enter the ATM switch (or AAL2 switch) will be VBR voice, it is likely to have bursty characteristics, and hence sufficient buffering will be required at the switch input, as explained above. When the peak traffic rate of an application within the switch fabric to the output link bandwidth ratio is small (1:20), small buffers are usually sufficient [27]. However, when the traffic peak to link ratio increases, larger buffers are required. Output buffer requirement for an ATM switch on which AAL2 multiplexing is implemented need not be very large. If the AAL2 traffic is

well scheduled onto the input link, the individual bursty effects of the traffic will be smoothed out and the input traffic will appear more as a CBR source to the switching fabric. This is why relatively small buffers are required.

If the input buffering for the switch is implemented as a single FIFO, head of line blocking is likely to occur. This is because both ATM and AAL2 cells will be present in the ATM stream, and only get separated once they enter the AAL2 processing module of the switch. Head of line blocking can be avoided by placing buffers on the output side, but this would require a high speed AAL2 processing module (essentially non-blocking) and the output to the AAL2 switch would only have the link capacity of one port of the switch to its disposal, instead of full access to the switch fabric. It is relatively simple to calculate buffer thresholds in an ATM switch to suit AAL2 requirements [5].

- **Routing** - Routing in ATM switches is typically either static or dynamic. Static routing implies that traffic will take a deterministic path through the switch. However, this often results in blocking as paths soon become congested. Instead dynamic routing allows traffic to pass through multiple paths through the switch, and hence traffic is evenly distributed through the various stages of the ATM switch. We shall later see how the WUGS switch itself exhibits a dynamic switching architecture.
- **Multi-casting** - Multi-casting is referred to as the ability of an ATM switch to copy the contents of a cell and instead of outputting the cell to a single port, outputting the cell to multiple individual ports, or a range of ports. Multi-casting is of not much value in bidirectional traffic flows, where interaction between end-users is commonly taking place. However, for voice distribution networks, multi-casting can be highly effective. Using multi-casting to broadcast ATM cells with AAL2 sub-channels has not been presented in the literature to date, but could play an important role when streaming a set of AAL2 sub-channels to customers, with each customer wishing to accept a subset or all such channels.
- **Scheduling** - Many scheduling algorithms for ATM switches have been proposed in the literature, and some of these algorithms emulate the preemptive scheduler that is found in many operating systems, which are responsible for scheduling tasks. As tasks in a system have various priorities, and ought to be scheduled according to their importance, so does ATM traffic. Real-time VBR voice should have a higher priority than data traffic, and the switch mechanisms should ensure that this voice

traffic reaches the AAL2 switching modules in time to avoid increases in delay or jitter. As the AAL2 module will be implemented as an input-side module, it will be necessary to investigate the effects of input scheduling schemes. We shall also see in Chapter 5 how scheduling is extended to the AAL2 switching module to allow for the prioritisation of real-time traffic flows.

University of Cape Town

Chapter 3

System Design for the AAL2 Switch

3.1 Introduction

As was stated before, the mechanisms of AAL2 allow data from multiple users, each user occupying a unique AAL2 channel, to flow along a single ATM connection. This is achieved by setting up a point-to-point AAL2 connection. A point-to-point connection implies a statically created connection between any two arbitrary points in the ATM network. At the source, compressed voice packets are arranged into 48-byte protocol data units (PDUs) before being passed to the ATM layer for delivery, and hence we refer to this process as a CPS-Packet to AAL2-PDU transformation, or simply AAL2 encoding. At the destination end, the inverse process occurs, where AAL2 PDUs are re-converted into the original CPS packets. We refer to this inverse process as an AAL2-PDU to CPS Packet transformation, or AAL2 decoding. If a direct path exists between these two nodes, i.e. no AAL2 switches are present, then one would expect the data on the path to undergo both these transformations only once.

If we shift our attention away from the point-to-point connection scenario, and focus on the AAL2 switch itself, we find an interesting phenomenon. As AAL2 switching may sometimes be regarded as a new virtual layer above the AAL2 CPS, i.e. at a packet level, the switch must first decode any AAL2 PDUs into CPS packets before they may be processed. Likewise, all outgoing packets must be re-converted, or be encoded into an AAL2 PDU format before leaving the switch. There is a fundamental difference between the point-to-point scenario just discussed and the switch scenario, as shown in Figure 3.1. In the switch, the order in which events occur, i.e. the order in which information

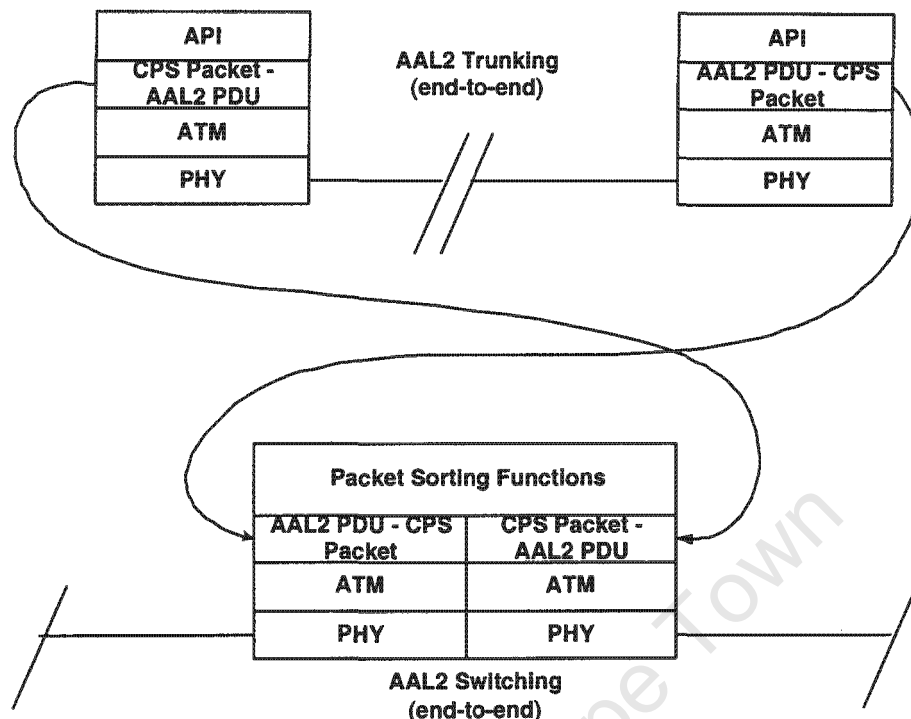


Figure 3.1: AAL2 Duality

is processed, is reversed to that of a point-to-point system. Whereas the transmission channel in the first instance was simply an ATM link carrying AAL2 PDUs, the so-called transmission medium in the switch would be a high-speed bus instead whose function would be to carry these CPS packets. This duality principle allows one to conclude that AAL2 switching essentially incorporates the same principles of encoding found on an AAL2 trunk, but simply with these mechanisms inverted, and that a further internal mechanism is required to provide switching or regrouping of the CPS packets to the correct outgoing stage.

In the preceding chapters we presented AAL2 as a suitable technology for the transport of packet-based voice, and motivated the necessity for the AAL2 switching node. An AAL2 switching node includes user, control and management functions to facilitate its primary task of switching, by enabling dynamic connection setup and synergy with the surrounding network (signalling) as well as network monitoring, emulation and accounting features (management functions). Therefore the word “node” should be taken to imply a switch in which the above features are present. Henceforth, when referring to the AAL2 switch alone (without the “node” suffix), we are referring to the user-plane model of the

switch. It is this aspect of the switch that these studies will focus on.

Before presenting a fully-fledged system design, we will examine the requirements of the AAL2 switching node in more detail, as these requirements will outline the system definition from which a suitable design may be forged. The system definition given in System Definition Language (SDL) will guide the reader as to how the system should achieve the various functions. These functions are further mapped to a series of modules, each of which operate independently from one another, but yet work co-operatively to achieve a fully-operational system.

Before proceeding any further, the reader should be aware of various terms that are synonymous with the AAL2 technology. An *AAL2 channel* may be regarded as a point-to-point unidirectional pipe that is capable of transporting voice traffic in packetised form. Each AAL2 channel is identified by its unique CID value. An *AAL2 path* is defined as a single ATM virtual connection that is capable of carrying multiple AAL2 channels. By capable, we imply that various signalling procedures would have occurred during the connection setup phase to create such a “channel aware” AAL2 path. The words path and channel will be used interchangeably throughout this document to mean AAL2 path and AAL2 channel, respectively. Finally, in discussions involving the AAL2 technology, mention shall be made of AAL2 PDUs, which have been packed according to the recommendations given in ITU 363.2, and non-AAL2 PDUs, which may represent any other type of PDU, including those representing AAL5 and AAL0 connections. The words “standard PDU” and “standard cell” will be used to indicate non-AAL2 type PDUs and non-AAL2 type cells, respectively.

Finally, a good architecture is dependent on a good switch design. A good design is not one in which the winning attribute is cost. As the switch is intended to form part of a core switching network, cost is of secondary importance. A good switch design is one that allows the internal mechanisms of the switch, even if implemented as higher layer protocols, to best respond to the traffic at hand, once again voice. A good design is one that scales well between serving tens of customers to thousands of customers.

3.2 Switching Node Functional Requirements

In the previous sections, we examined the requirements for an AAL2 layer to carry compressed voice, and the need for AAL2 switching, but with emphasis on economic feasibility and efficiency gains. In this section, the reader is presented with a technical overview of the

various system requirements. These requirements have been selectively placed into three groups for clarity. In the first group, we examine system-wide or global requirements that should be addressed by the switching node as a whole. The second group deals with those requirements that ought to be met exclusively by the AAL2 layer mechanism. The final group will take a look at switching requirements at the CPS packet level.

The following requirements relate to the global switching system:

- To support a dynamic system, the use of AAL2 signalling is required. AAL2 signalling requests are conveyed to the system at the AAL2 layer and not at the ATM layer, and signalling messages are carried in the payloads of CPS packets. This signalling would be responsible for the set up of and tear down of AAL2 channels within an AAL2 path. More importantly, AAL2 signalling will be responsible for the maintenance of the internal CID translation table. It is therefore a requirement for the system to be able to reroute signalling information to the appropriate sub-systems, that may then deal with these issues.
- The AAL2 switch must not only support the switching of AAL2 cells, but standard ATM cells as well. It should be the task of ATM signalling to route the AAL2 and ATM flows to independent switch inputs, but in the absence of signalling, a more sophisticated probing mechanism may be required. Note that although AAL2 signalling is being taken into account, it does not form the focus of this study - the primary emphasis will lie on achieving functional AAL2 switching.
- The system should be able to support logging for accounting purposes, and traffic flow information for evaluation purposes, which may both be implemented as management functions.
- The switch should support multiple paths for two reasons. Firstly, it is likely that under heavy load more than 248 channels would be required on the same end-to-end route. For this reason, multiple paths may be used in parallel to support more channels than the ITU imposed limit of 248 per path, in a similar fashion to tandem trunks found in many public voice-carrier networks. In addition to this, for each separate end-to-end route, a separate AAL2 path is required. The switch must support these multiple paths in both the input and output stages.
- As the input stage receives CPS packets from each incoming path, it must be able to keep track of what position in the PDU the following packet is to be retrieved and

whether any further information can in fact be extracted at the current time. It may therefore be said that a requirement of the system is to maintain the correct state for each input path, so that when more information arrives, processing may continue where it left off. It should also be evident that the state of a particular AAL2 input path is unlikely to be the same as that of any other path, and hence the system is required to hold the corresponding state information for each input path that exists.

- Similar to the input paths that have been discussed, each output path will also be characterised by its own state variables, and these must be maintained.

The following requirements relate to the standard operation of a typical AAL2-layer mechanism, without taking switching into account:

- For each AAL2 path being administered by the switch, the channels comprising the path should be extracted in time for switching purposes. As soon as it is clear to the system to what output port a channel must be switched, the information on the channel must once again be formed into the suitable protocol data units for transmission.
- For each AAL2 path, the switch should conform to the end-to-end delay requirement to support the requirements of voice traffic. To avoid having to incorporate echo cancellation equipment into the network, this delay should be no greater than 30 ms.
- The system should support up to 248 channels per path, and channel 8 on each path should be reserved for signalling purposes.

The following requirements relate to that of AAL2 switching:

- Suitable queues should be provided to hold pending voice packets. There are two stages in the switch where a voice packet may incur blocking. The first block may occur before the scheduler, where a packet must wait to be scheduled. The second block may occur where a packet is about to be inserted into an outgoing PDU, but cannot, as the output AAL2 mechanism is not yet ready, and the packet is delayed somewhat. Queues hold these packets as they build up to insure packet loss is kept within a reasonable limit under heavy load.

- An internal table should be available for switching packets to output paths based on three labels, namely the input path on which the channel entered the switch, the channel ID itself (CID) as well as the output path to which the channel should be assigned. By using these first two parameters as indices into a lookup table, functional switching may be achieved. An important organisational requirement that will impact on the system design is that the entries in this table need to be accessible by sub-systems that will provide dynamic updating of the switch's internal environment.
- It was stated earlier that the use of voice compression imposes varying delay constraints on each network element, in this instance the AAL2 switch, and that these delay constraints are typically dominated by the codec in use for the particular channel. It was also stated that the more powerful the compression scheme used, the more sensitive the application would be to delays. Each channel in the switch will therefore represent voice information (or data) that should be treated with a suitable quality of service that must in some way match the voice delay requirements. This translates into the need for scheduling within the switch to support these varying service types.

3.3 System Definition

Having established these three groups of requirements, it is now possible to examine the actions that ought to be taken in the processing of a single protocol data unit, as it enters the system. These actions are shown in the SDL diagram of Figure 3.2. A description of SDL repertoire can be found in Appendix B. The reason for only considering a single input and output path in this diagram is that the steps taken in processing packets from multiple paths are identical. The only difference between these paths is that they must each maintain their own state. It is not possible to show current state information in the system definition below. We shall therefore assume, for demonstration purposes, that the internal CID table has been set up appropriately so that all channels extracted from the single input path happen to map directly onto a single outgoing path.

The operation of the system to process a single PDU may be given as follows:

- When the system is in the IDLE state, it has completed all pending work, and is awaiting a fresh PDU from the ATM layer below for processing.

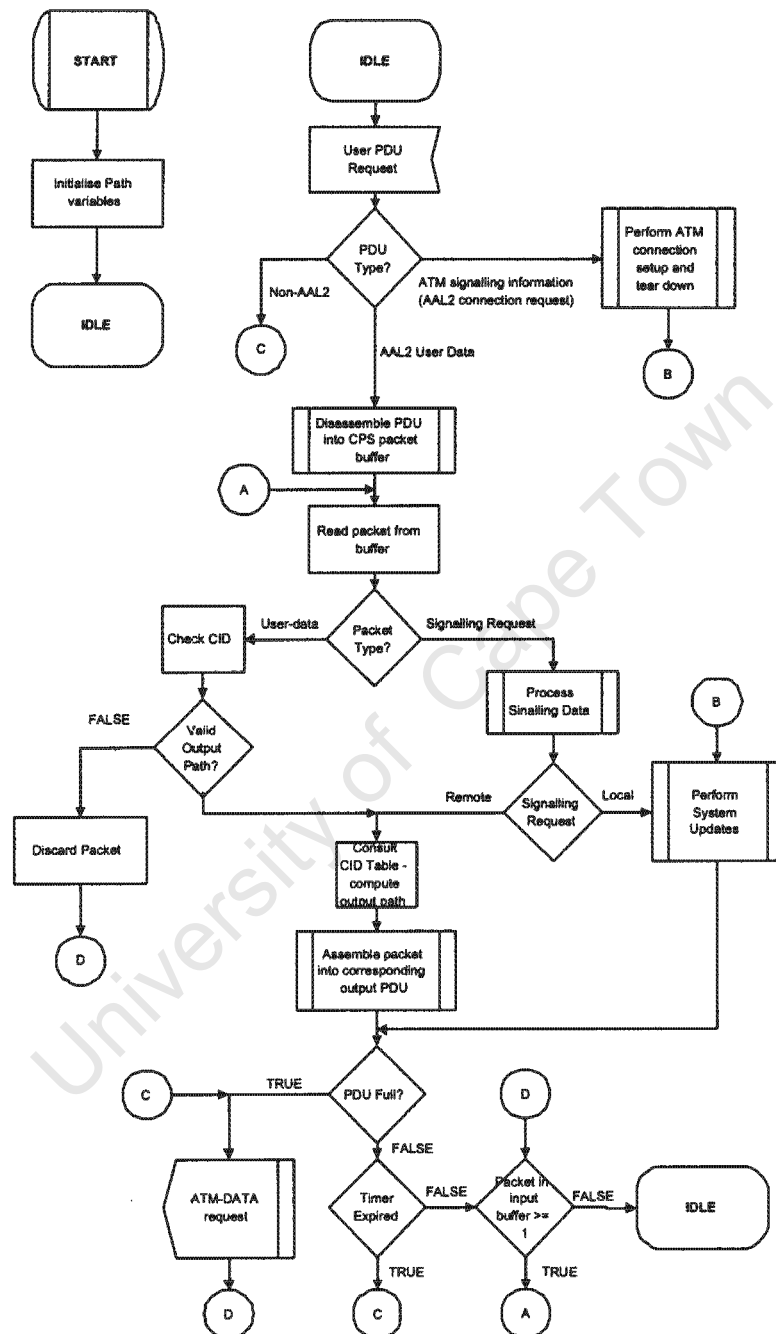


Figure 3.2: SDL Representation of AAL2 Switching

- Once a PDU has been successfully received, a decision must be made before it may be processed any further. If the PDU is a standard AAL PDU it must be passed to the ATM layer for VPI/VCI switching. It should be noted that the SDL diagram of Figure 3.2 is not specific to either the user-plane or control-plane and may in fact encompass both. Therefore, the possibility exists that the incoming PDU may contain ATM signalling information. If the PDU does contain any valid signalling requests, these are processed, and any required system updates are performed. An example of such an update would be a request to allocate a new AAL2 path.
- Although non-AAL2 PDUs have been catered for, the incoming PDU is most likely to contain AAL2 user information instead. Therefore the next stage would most likely involve extracting as many packets as possible from the PDU, and placing these complete CPS packets into an array of buffers. A separate buffer is to be allocated for each individual channel that exists, including channels reserved for signalling. Note that although we are currently concerned with a single AAL2 path, in the case when multiple paths exist, and a channel CID is duplicated, this will result in multiple buffers being allocated for these channels. The allocation of these buffers depends on how many paths carry the channel with the same CID. It is therefore important to note that although two particular channels may have the same CID value, if they don't belong to the same path, they are not alike, and should be treated as unique channels.

Once again, the need to retain state information for each incoming path cannot be overemphasised. The first few bytes retrieved from a PDU that represent valid data may mark the beginning of a new CPS packet, or a continuation of a packet already under construction from the previously received PDU. If the latter case, had the system not been aware of the progress of the packet at the end of the previous PDU, errors would result when trying to process "meaningless" data from the new PDU. The fact that the system is able to retain the per-path state makes this packet retrieval mechanism possible.

- A loop is now entered at C in Figure 3.2, where successive packets are retrieved from the input buffers and processed. The number of packets processed before the loop expires, with the proviso that packets are always available, will depend on the scheduling policy selected to provide the required quality of service to the channel bound to the packet stream.

- Once a packet has been read in from a buffer, the header is examined. If the packet contains information that may be classified as signalling, the packet is submitted for processing by a signalling entity. This signalling entity is capable of accepting packets with signalling commands or information and processing this information. It then makes decisions based on the information, and carries these out in the form of updates to the system and the surrounding network environment by generating either events or signalling messages. If a local update is required (i.e. within the switch system itself) the signalling entity is capable of triggering other sub-systems to carry out the relevant commands, e.g. adding a channel to an incoming path. If a remote update is required (i.e. to a distant AAL2 switch) the signalling information is encapsulated into one or more CPS packets, and the CPS packets are submitted for translation on a valid output path.
- If the packet header indicates that the packet contains user data, a further check is applied to ensure that the packet belongs to an already allocated channel corresponding to the input path on which the packet entered the system. If no such match exists, the packet is simply discarded. If the packet is valid, it may be submitted to the scheduler.
- In the scheduling stage, packets that belong to the same scheduling group are taken from the input buffers and translated to the output buffers, in a manner that is governed by the scheduling mechanism assigned to the group. Once scheduling of a group is complete, the following unprocessed group will receive attention, and so on, until all input buffers with one or more packets in them have been accessed at least once. (The more times a particular buffer is accessed of course depends on the scheduling scheme)
- At the output stage, the CID table is examined, and a path match is found. Successive bytes comprising the packet are then copied into the remaining space of the PDU currently under construction.
- When no more space is available inside the current PDU, it is ready for transmission and may be dispatched. A new PDU is then created and the outstanding packet contents are placed inside this new PDU.
- Once the processing of the current packet is complete, the system will continue the loop if any packets should still be processed, or else return to the IDLE state once

again, where more information may be received.

- For each outgoing AAL2 path that is created, a separate timer is responsible to ensure each PDU is submitted to the ATM layer on time. The timer value forms yet another state parameter of each outgoing path. When a timer expires, the PDU under construction is filled with padding before being transmitted.

3.4 System Design

To support AAL2 switching, we realise the switch design may be broken up into two main logical segments. These segments are logical in the sense that they do not map directly to any discrete system modules. A segment may in fact comprise three or four modules to operate correctly. The first segment incorporates the mechanisms to support the breaking up of AAL2 PDUs into CPS packets and vice versa, and the second segment comprises the mechanisms to support the core CID switching function. At this stage, switching implies the translation of CPS packets to the correct output path based on the translation table, and does yet take into account any scheduling mechanisms. To function correctly these systems require the support of various sub-systems distributed throughout the switch. These sub-systems include but are not limited to signalling components, that take care of updating information dynamically and queue managers that are responsible for dynamically allocating memory for buffers on demand.

As the operation of the system at runtime will be highly asynchronous, a sensible approach is to ensure that tasks that may be grouped together and form one aggregate function are grouped into their own unit, and that each unit operates independently from others. Although a unit may require data from a neighbouring unit to function, it is not aware of the state of its neighbour, and cannot predict when new data will become available. We may regard the AAL2 switch as a non-symmetrical system for the following reasons:

- The quality of service required for each channel will place separate requirements on the internal scheduler at different times. This will affect the time taken to schedule batches of packets that may belong to the same QoS group as well as the rate at which input and output buffers are emptied or filled (which will of course affect the operation of neighbouring modules)

- As the number of input and output paths increases, the number of state machine instances in the system will increase, i.e. the state machines that control the segmentation and re-assembly of AAL2 packets. When control is passed to a state-machine instance, it is not trivial to determine the time that will elapse until processing is complete, and this places further uncertainty on the operation of the system.
- Each path may carry any number of connections from 1 to 248, and the ratio of signalling requests to user data on a given path may vary. As signalling requests are treated with higher priority, user-classified data will have to wait until these requests have been processed.

The overall design diagram is given in figure 3.3, and consists of nine separate units, which may be described as follows:

- **The Admission Control Unit (ACU)** is responsible for receiving ATM-level signalling requests on a well-known address. These UNI-based requests are intended for the establishment of AAL2 connections. If a request is received to allocate a new AAL2 path, the SPU (discussed shortly) is notified to create this path. At this state the ACU is a null entity and the establishment of AAL2 paths is assumed to be the responsibility of a management entity.
- **The Interrogator Unit (IU)** is required where ATM signalling has not been implemented, or is not possible. ATM signalling incorporates connection setup mechanisms, which allow the ATM VC to be classified as a carrier for AAL2-type traffic. In the absence of ATM signalling, ATM cells of any type may enter the switch on the same connection identifier (i.e. the VPI/VCI index). It is the responsibility of the interrogator unit to examine the payload of such cells carefully to identify whether the given cell payload contains AAL2 type data or not. The IU would perform such a function by scanning the payloads of cells to determine whether any distinct patterns may be found. As an example, the CID field of an AAL2 cell can only take on certain values¹. If, in an unknown cell, the field in which a CID would exist, is found to have a value of 1, then the cell couldn't possibly be an AAL2 cell. Of course, only one such clue would not necessarily be sufficient to guarantee the type of a cell. Such a function would most likely be processor intensive, and would best be implemented in hardware.

¹Values of 0-7 for a CID are not allowed and have been reserved.

AAL2 Switch - Modular Design Diagram

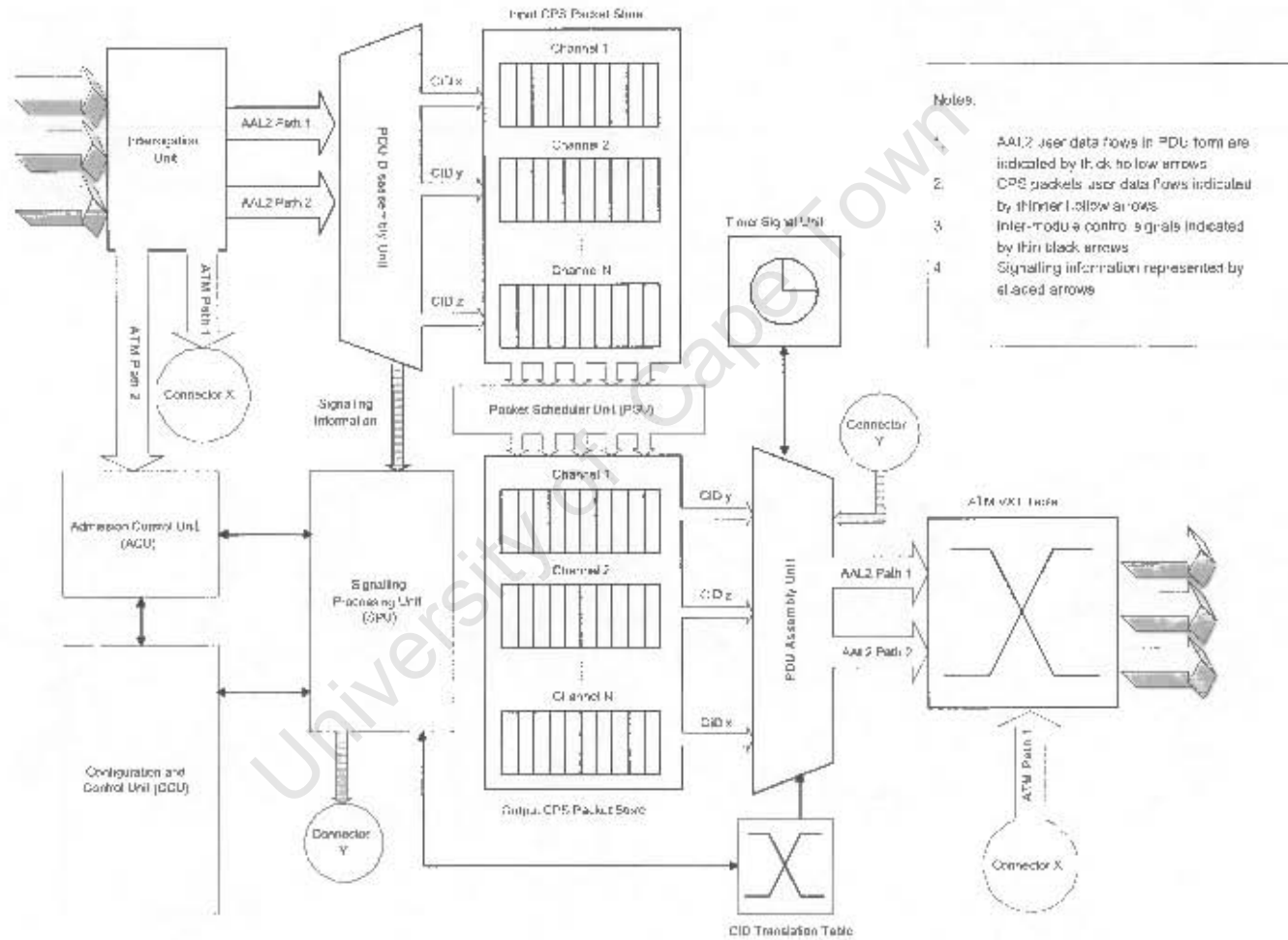


Figure 3.3: Switching Node System Design

- **The AAL2 Disassembly Unit (ADU)** provides the mechanisms for converting PDUs into corresponding CPS packets, and placing each packet into its associated buffer for processing. The ADU must be capable of handling multiple input AAL2 paths and retain the correct state variables for each input path.
- **The Input and Output CPS Packet Stores** hold packets that are waiting to be scheduled or reassembled. These stores are accessible to the SPU, ADU and ARU as these units either create, insert or remove packets respectively.
- **The Packet Scheduler Unit (PSU)** copies packets from the Input Packet Store to the Output Packet store. It differentiates between different channel QoS requirements by grouping equal or similar QoS's into a single type of service, and performing a suitable scheduling algorithm on the type of service group.
- **The AAL2 Re-assembly Unit (ARU)** reads in the CPS packets from the Output CPS Packet Store. The unit has access to the CID translation table. It performs a lookup for each packet to determine the outgoing path². Once a path is found, a mechanism takes over the role of reconstructing the PDU for that path. This unit may also be interrupted at any stage by the Timer Unit, in which case the path targeted will have its next outgoing PDU immediately dispatched.
- **The Timer Unit (TU)** has the highest priority of execution in the system. This unit is responsible for maintaining the timer-CU for each outgoing path. The TU is able to interrupt the system at any stage and pass control to the ARU. The ARU will take over the task of padding the PDU and transmit it, when necessary, as discussed above.
- **The Signalling Processing Unit (SPU)** takes care of signalling requests it they arrive, and makes decisions based on the information to perform dynamic updates to the system. It is primarily responsible for creating end-to-end routes through the switch, i.e. a mapping from input path and channel to a corresponding output path, and the tearing down of these connections. When a new channel is created, the SPU will allocate both an input queue and output queue in the relevant stores to accommodate packets of the channel. When a new route is established, the SPU will determine whether an outgoing path exists to accommodate the route, and if not,

²Note we are referring to a logical lookup. We shall examine the actual mechanism for performing a lookup in Chapter 5

create a new output path. Furthermore, it will place the relevant entry in the CID table, to be used by the ARU in its re-assembly process. The SPU must maintain the current active input and output paths, as well as the CID table. It is also the function of the SPU to remove routes, channels and paths when they are no longer needed, and to reject requests that may bring about unwanted effects in the system, e.g. a user trying to remove a path with one or more active channels.

- **The Configuration and Control Unit (CCU)** is responsible for configuration and system startup. The CCU will initialise all the various modules of the system on startup, and ensure that the relevant time elapses for the system to reach steady state, before accepting new data.

We have yet to examine how these modules will interface to one another at the system level, i.e. the interface signals that are required between modules in the system. The signals are the following:

- **VALID_IN_SIG_PACKET** is initiated by the ADU to inform the SPU that a packet containing AAL2 signalling information is available in an input queue for processing.
- **VALID_OUT_SIG_PACKET** is generated by the SPU to the ARU when a packet containing a signalling request destined for a remote entity must be transmitted
- **ADD_CID_MAPPING** is a signal sent from the SPU to the CID table entity to create a new Input Path / CID / Output Path Mapping
- **START_TIMER_CU** is sent by the ARU to the TU when the construction of a new PDU is commenced.
- **RESET_TIMER_CU** indicates that the current timer is no longer required (the ARU has either sent the PDU as it was filled before the timer expired or the construction of the PDU was cancelled)
- **TIMER_CU_EXPIRED** is generated when a timer reaches its limit and the ARU is required to immediately dispatch the current PDU. The signal is sent directly from the TM to the ARU.

- **ALLOCATE_CHANNEL_QUEUE** is sent from the SPU when a new channel is being set up, to pre-allocate the input and output queue resources that are required to buffer packets for the channel. This signal is targeted to two units, namely the Input Cell Store and Output Cell Store
- **FREE_CHANNEL_QUEUE** frees buffers for future channels.
- **ADD_AAL2_PATH** is sent either by the ACM or CCU module to the SPU to dynamically or manually set up a new AAL2 input path
- **ADD_AAL2_CHANNEL** is sent by the CCU to manually add a channel to a given path
- **ADD_AAL2_ROUTE** may also be initiated by the CCU to add an end-to-end route for a channel.

Chapter 4

WUGS Architecture

4.1 Introduction

Before the implementation aspects of the AAL2 switch can be discussed, based on the design from the previous chapter, it will be essential at this stage to examine the WUGS architecture on which the switch is based. It could be said that the WUGS switch and supporting environment to date offers the most extensive research capabilities in the active networking arena, by allowing one to effectively “cut-through” the physical technology barrier, and instead concentrate at the higher level system design and accompanying protocols.

When the Washington University Gigabit Network Distribution Programme commenced in 1999, the university in the position to ship their base ATM switch, i.e. the Washington University Gigabit Switch (WUGS) to selected participants (including the University of Cape Town), thus paving the first steps towards the research the programme had intended to promote. The standard kit was comprised of the following components:

- The base ATM switch
- ATM Port Interconnect Chip Network Interface Cards (APIC NIC), to allow end stations to communicate with the switch over an optical link
- Line card adapters, to form the transmission interface for each switch port, and hence provide the relevant optical to electrical conversion, at two possible link speeds

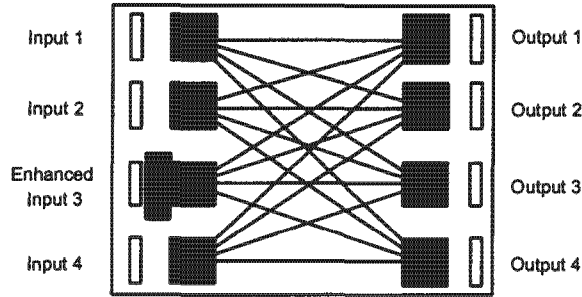


Figure 4.1: Enhanced WUGS Port Functionality

Since 1999, two additional components have been released. Both the Smart Port Card (SPC) and Field Programmable Port Extender (FPX) allow high-degree, per-port customisation of the switch, depending on what port these modules are located. Naturally the overall power and degree of functionality these modules can provide scales according to the total number of occupied switch ports. By “occupied port”, the author simply implies a port that has been extended with either of the above-mentioned components. Although the FPX module is to be targeted in the following stage of this research, it is currently out of scope of this text, and the reader is instead referred to the recommendations in Chapter 8.

This chapter will introduce and describe the various WUGS components that make up this environment, and the relevance of these components to the implementation of the AAL2 switch. The reader is cautioned of the abridged manner in which these components are described, as it is simply not possible (and not relevant) to provide the same detailed coverage that can be found in the supporting literature from Washington University [10]. The function of the AAL2 switching module, in extending the WUGS architecture is shown in Figure 4.1 above.

Figure 4.1 makes the clear distinction between the AAL2 switch design domain, and the WUGS environment domain. The objective of this chapter is therefore to focus on the supporting WUGS environment that will make the later implementation possible. The following chapter shall then present more close-up coverage of the actual implementation specifics.

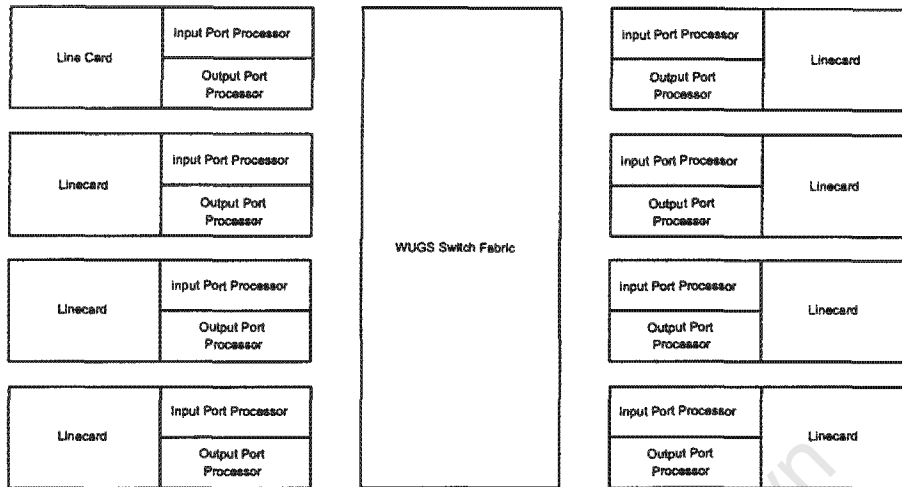


Figure 4.2: WUGS Internal Organisation

4.2 WUGS Base Switch Overview

The Washington University Gigabit Switch or WUGS is an 8-port ATM switch, the internal organisation of which is shown in Figure 4.2. The architecture of the switch allows for the number of ports to be extended to 64. In order to provide connectivity from an optical medium to the switch port processors, two separate linecards are distributed. A low-speed dual OC-3 linecard allows per-port throughput of 310 Mbit/s (2 x 155 Mbit/s). A linecard capable of operating at a much higher speed of 1.2 Gb/s, with proprietary optical technology from Hewlett Packard, is also available and was subsequently used for each AAL2-enabled port of the switch. Although the throughput provided by the original OC-3 linecard would have sufficed for the purpose of this research, this linecard is not compatible with the APIC NIC, which itself also runs at a link speed of 1.2Gbit/s, and was the only ATM-based adapter currently supported by the version of NetBSD at the time. Commercial ATM adapters that do operate at OC-3 link rates generally do not have open-source specifications and are hence ill-supported on UNIX-based platforms.

The switching fabric forms the core of the switch and connects each input port processor to all possible output port processors. The fabric is based on a well-known, multi-stage switching architecture, known as a Benes switching architecture that contains multiple stages or ranks of individual switching elements. Each of these switching elements is in turn configured as a simple single stage crossbar interconnect. As demonstrated in [28],

the number of stages in a Benes switching scheme may be expressed as $2k + 1$. The first $k - 1$ stages distribute the incoming load as evenly as possible, which the last k stages route individual cells to their corresponding outputs. In this way, switch contention is greatly reduced as the internal throughput of the switch need be no higher than that of any external link [28].

To reduce the internal speed at which the switch must operate, four parallel planes are provided. Data is transmitted through the switch as consecutive 36-byte words, each containing a single four byte header and four additional bytes, each passing through a separate switch plane [5]. The internal switch fabric is synchronously clocked at 120 Mhz, and fourteen cycles are required to transmit an ATM cell through the switch fabric.

In order to perform cell header lookups, a lookup table known as the Virtual Circuit Translation table (VXT table) is provided that can hold a maximum of 768 simultaneous entries [5]. To make modification to this table, control cells must be sent to any input port of the switch. A control cell is simply a raw ATM cell (AAL0 cell) with its VC field set to 32. Depending on the orientation of various fields in the cell payload and their values, the VXT table will be updated accordingly. To simplify the process of controlling the switch by having to send individual cells, two utilities are provided. The first utility, called GBNSC, accepts commands from a remote application over a TCP socket and encapsulates the command into the control cell payload in a form that can be understood and carried out by the WUGS. The remote application that sends the commands to GBNSC is known as Jammer, and can either read a control script from disk, or accept individual commands from a user in an interactive session. The use of these tools is documented in Appendix E.

4.3 APIC NIC

The ATM Port Interconnect Chip (APIC) is a high-performance network interface chip developed by Washington University for the sending and receiving of ATM traffic at a maximum speed of 1.2 Gbit/s. To allow the APIC to interface directly with a system's main memory, it contains a PCI compliant IO bus that operates at a frequency of 33Mhz. As all the APIC registers can be accessed directly from the operating system and are open-sourced, a high level of control and customisation can be achieved.

To allow the APIC to operate as a network interface on a standard PC-based workstation, the APIC chip was integrated with supporting components to form the APIC network

interface card (NIC). The APIC NIC plugs directly into an available PCI slot on any system motherboard. A simple kernel upgrade is all that required to allow the APIC to be accessible to the system. There are two ways in which the APIC may be controlled from software. In the native socket mode, raw ATM sockets are bound to a particular virtual connection on the APIC, allowing special 56-byte cells to be sent to and received from the APIC.

One of the drawbacks of the native socket interface is the inability to control the APIC from user-space, i.e. from applications that are run by a so-called standard user of the system. Another disadvantage of sockets is the limited efficiency that is achievable, due to the fact that all accesses to the device must occur via a software-based device driver. To alleviate some of these drawbacks, a special set of usermode libraries have been provided by Washington University that provide direct access and control to the on-chip registers, hence providing an additional means by which the chip may be controlled. This results in better network efficiency, particularly when transporting large higher-level data frames, such as AAL5 frames. The main drawback of the usermode libraries though, is their complexity and low-level knowledge of the APIC chip that is required. The socket interface for the APIC is considerably simpler and far more abstract than the usermode libraries. Furthermore, although the usermode libraries do provide better efficiency when dealing with longer frames, transmitting smaller frames or individual AAL0 / AAL2 cells is not as efficient. It was also felt that the efficiency gained from the usermode code would be offset to some extent by the limited speed of the SPC, and hence the throughput that could be provided¹. For these reasons, it was chosen to use the native socket interface of the APIC in the interim, allowing more focus to be placed on the software adaptation layer functionality of AAL2 instead.

As AAL5 is undoubtedly the most popular manner in which to transport applications' traffic over an ATM network nowadays, the APIC can operate in two modes. The most common is the AAL5 mode, in which AAL5 frames are segmented into individual ATM cells and recovered at the receiving end. In the AAL0 mode, raw ATM cells or buffers of raw cells are passed directly to the APIC for transmission, and hence there is no presence of an ATM adaptation layer. Therefore there is no support for AAL2, and all AAL2 packet encapsulation must be performed in software. In fact, the AAL0 mode of the APIC is recommended for implementation of non-supported adaptation layers [7]. As explained

¹The operation of the AAL2 modules are significantly more complex than traditional switching, with contributes to additional processor cycles on the SPC being consumed.

in chapter 2, packets arrive at this software adaptation layer and are placed into 48-byte PDUs. Upon completion of the PDU, 2 headers are appended to form a 56 byte cell, which is then directly passed in the AAL0 mode to the APIC for transmission. The first header is simply the ATM cell header without the HEC field, which will be appended by hardware before the cell is dispatched. The second four byte header is an internal APIC header, which is used for routing purposes.

The APIC chip contains three individual ports [7]. The first port is the input port on which the APIC receives ATM cells. All constructed cells are sent out on the second port or output port of the chip. Furthermore, the APIC communicates with the main system via the PCI bus interface, which may be regarded as the third APIC port. This leads to three paths that cells may take through the APIC chip. The receive path is defined to be the path from the input port of the APIC to the system bus interface. The transmit path is the path taken by cells that originate in main memory to be transmitted onto the ATM link and which should exit through the output port. Finally the transit path applies to cells that simply enter the APIC through the input port, and pass directly through to the output port. If the APIC has not been explicitly configured to accept data from an input port on a specified VC, the default action taken by the APIC is simply to propagate the cells from input to output.

4.4 The Smart Port Card

The Smart Port Card (SPC) is an embedded computer capable of performing cell-level processing in software. The SPC is based on a Pentium embedded module running at 166 MHz with on-board level-2 cache and a Northbridge controller to coordinate activity between main memory, the CPU and PCI bus. A custom FPGA has been integrated onto the SPC-board to provide system BIOS capabilities and synchronisation in the form of software interrupts to the operating system. As the SPC resides in the WUGS, there is naturally no need to support standard peripherals such as keyboards, mice, screens and the like. The only access to the SPC is provided via two serial ports (to provide shell access to a user) and the APIC interface.

The APIC interface allows the SPC to receive and transmit ATM cells to and from the switch ports and core switch fabric. In the case of the SPC, the APIC chip is integrated directly on-board with the existing SPC components and communicates directly to the

PCI bus. In the previous section, the three ports of the APIC were discussed. The three ports are identical on the SPC-integrated APIC.

However, there is one subtle difference between the APIC on board the SPC and the NIC-based APIC. Each APIC chip is connected to two physical interfaces. When the APIC is part of a Network Interface Card, the two available interfaces are a fibre-optic interface and a high-speed ribbon interface, which was intended to provide a cheaper alternative. With regard to the APIC on the SPC, instead of the fibre and ribbon port, one refers instead to the link and switch port, as the APIC on the SPC is directly “sandwiched” between the linecard of the particular port (link) and the switch fabric (switch). This is one of the reasons why an additional internal 4-byte APIC header must be appended to cells in software - this header contains routing information that informs the APIC whether the cell ought to exit on the fibre or ribbon port (NIC) or link or switch ports (SPC) instead.

To initialise the SPC, a kernel and corresponding file system are first prepared at a remote development workstation. The kernel need only be customised to support the bare essentials that would allow the SPC to achieve the correct tasks, e.g. APIC support and ramdisk support². Once complete, the kernel and file system are combined into a single image and downloaded over the ATM network to the SPC. Once downloaded, a sequence of bits provides an instruction to the SPC to start booting. Once booted, a user or kernel level process may take control. The process of configuring the SPC is discussed in more detail in Appendix E.

4.5 NetBSD Operating Environment

NetBSD version 1.4.1 was chosen by the Applied Research Lab [10]. It was released under the BSD license and is a free UNIX-based OS. A proprietary embedded OS obtained from a commercial vendor would undoubtedly have clashed with the Gigabit programme ideals. NetBSD was developed using the C programming language (as are most UNIX-based OS's) and is supported by almost all GNU applications, debuggers and tools, which themselves provide outstanding developer support.

NetBSD was felt to be a suitable OS as it provided several mechanisms required by the AAL2 switch module. Descriptors and socket support are required for file (logging) and

²Support for SCSI, RAID, video controllers, etc. which appear in standard NetBSD kernels is, of course, removed.

network access. Software interrupts are required to allow the timer-CU to operate correctly when transmitting AAL2 cells. As NetBSD is a lightweight operating system, it does not come bundled with too many generic (and not particularly useful) utilities and drivers. This is seen as an added advantage, as the SPC does not contain any secondary storage, and must rely on a ramdisk in main memory to store the file system and kernel³. The size of the ramdisk is further limited to 24 MB.

NetBSD does however, have its disadvantages. As already mentioned, it is not inherently a real-time operating system. This makes it not possible to provide strict QoS guarantees, particularly for voice traffic, especially under heavy loads. Voice traffic is sensitive to network delay and jitter, and a non-real time OS is only capable of offering relative priority to varying traffic grades. Secondly, due to the same reasons, the correct timing of the timer-CU mechanism cannot be guaranteed.

An additional disadvantage of NetBSD is the lack of support of threads, which themselves may be regarded as lightweight processes. However, this problem was eliminated by making use of a process-specific threading environment.

4.6 Integrating the WUGS Components with the AAL2 Switch

The preceding sections ought to have provided sufficient grounding to allow one to now examine how the various WUGS components may be used as a support structure for the AAL2 switch. It should once again be stressed that any AAL2 functionality is to be kept within the domain of the SPC itself. It should also be noted that as no signalling support is enabled in the WUGS, all VXT tables entries must be allocated manually.

For each port on the WUGS that the AAL2 module (hence SPC) resides, the corresponding port will be enhanced with AAL2 switching. Therefore, as there are eight ports on the WUGS, there naturally exists the opportunity of enhancing all eight of these ports. We shall however, in the example below, focus on two AAL2 enabled ports. The SPC is capable of processing both ingress traffic to the WUGS as well as egress traffic. There are two advantages of configuring the SPC to process ingress traffic:

³A symbol-only version of the kernel is stored on the file system to save space, and links directly to the already running kernel.

- The SPC has the throughput capability of the entire switching fabric, with the result that output queues on the AAL2 switching module need not be very large.
- Once cells have been processed by the AAL2 module, they will still need to be switched at the ATM level by the WUGS. It is simpler to achieve switching if the SPC is located at the input port - cells that exit are already at the input of the switching fabric, and are treated as conventional ATM cells. Had the SPC been located at the output port, the corresponding output port would need to identify these cells and recycle them through the switch, if a cell lookup was required. As the cell arrives on the corresponding input port, an entry in the VXT table would need to exist to ensure that the cell would be switched correctly in its second pass through the switch.

Having decided to place the AAL2 modules on the input ports of the switch, the following steps must be taken to incorporate the design onto the WUGS:

- All end-stations, sources and/or base stations that will be communicating with the WUGS are first identified. VCs are allocated for any ingress and egress traffic external to the WUGS and are noted⁴. If a source generates AAL2 traffic, it will link up to an AAL2 enabled port. Sources generating ATM traffic, or a mixture of ATM and AAL2 traffic that must terminate at a single port, are linked up to non-capable AAL2 input ports, where the connections will be switched to the appropriate entities (either output port if ATM or recycled back to input for AAL2).
- The input ports on which the AAL2 modules (SPCs) will reside must then be determined. These input ports should preferably not be used for standard ATM traffic, in order to provide the maximum bandwidth available to the SPC.
- If any ATM switches precede AAL2 switches, the ATM traffic may be passed through the WUGS to a recycling port, where the traffic is subsequently recycled back to an AAL2 module for processing.
- Any standard ATM connections that are simply switched from an input port to an output port should be entered into the VXT table.

⁴Not all VCs are directly entered into the VXT table.

- As AAL2 modules generate ATM cells, the VCs on which cells are transferred must also be included into the VXT table to ensure the traffic will propagate through the switch.

When an SPC generates APIC cells (56-byte cells), it is important to ensure that these cells enter the switch fabric and do not pass back out the input port. Therefore any output connections on which the APIC will send cells, must be configured to transmit cells on the switch interface and not the link interface.

In Figure 4.3, a testbed is shown with 5 base stations that either send or receive AAL2 traffic. Not all sending base stations connect directly to an AAL2 switch, but may be passed through an ATM switch to separate the traffic flows. Although there are 2 ATM switches and 2 AAL2 switches in the diagram, it is a simple matter to show how the entire functionality of Figure 4.3 may be included into a single WUGS.

The chapter concludes by providing a practical solution for realising a testbed that consists of both ATM and AAL2 switches, which is shown in Figure 4.4. Note how a separate VC is allocated for each ATM connection. Connections may carry one, two or three AAL2 channels simultaneously. Cells that enter SPC modules are likely to have their contents altered, and hence it would not be a good idea to maintain the same set of VC identifiers at the SPC output as the input. Note that the solution is not the most compact that may be realised, but has been deliberately spread over the entire switch to aid in simplicity.

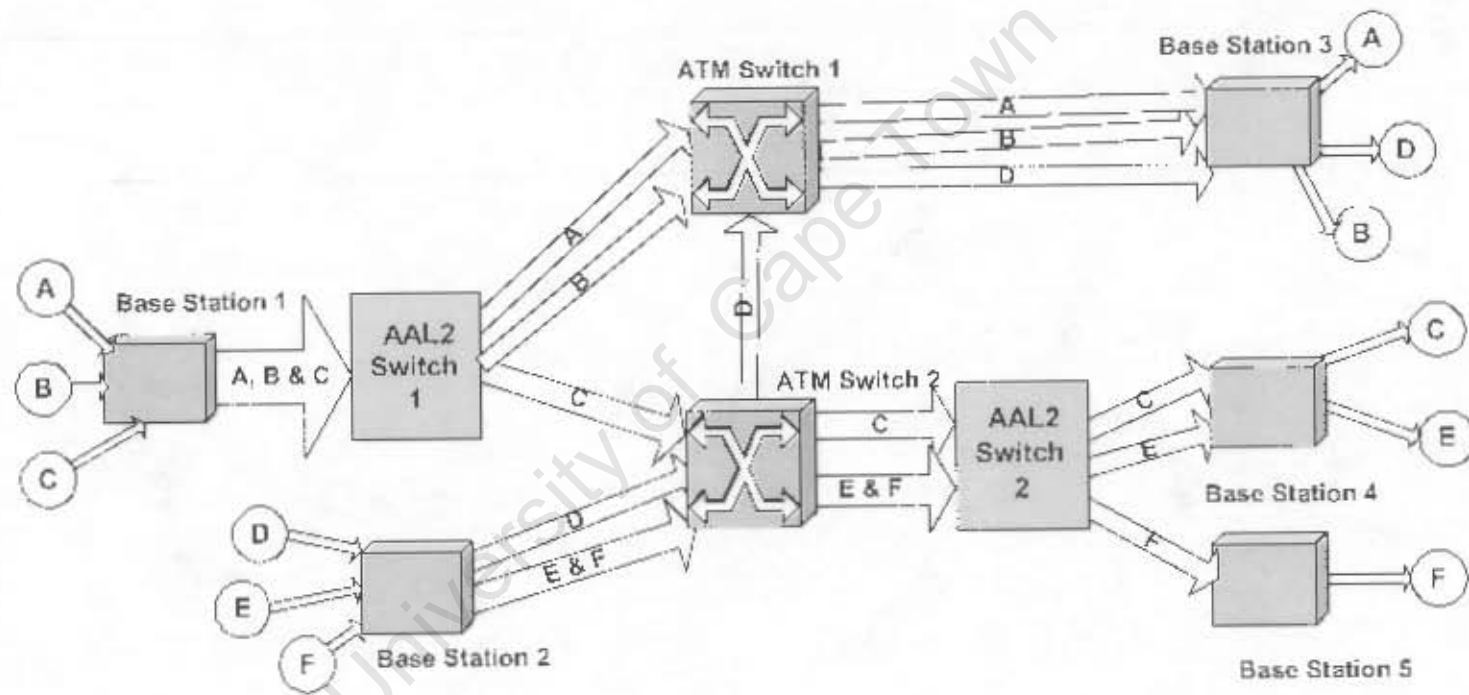
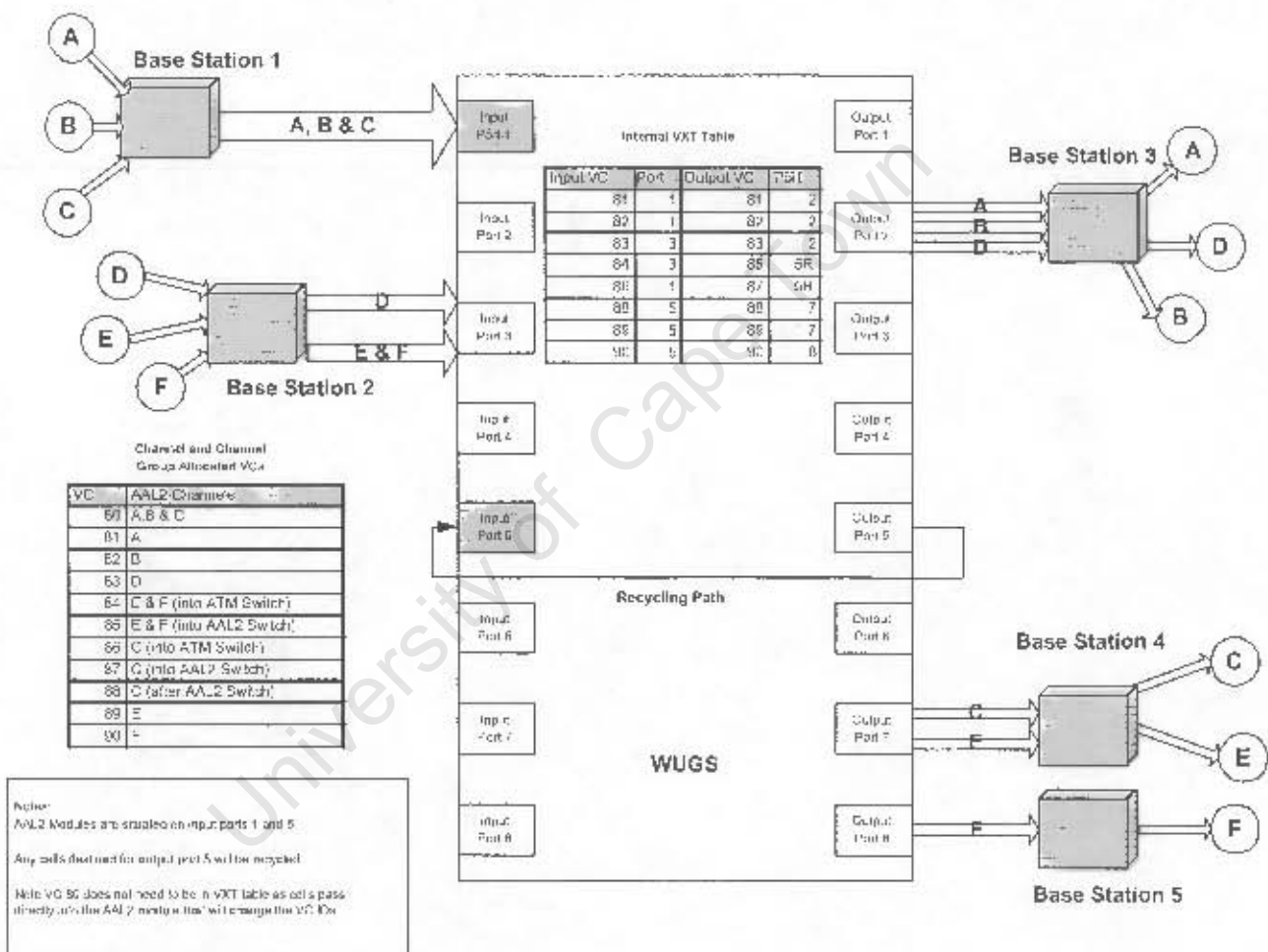


Figure 4.3: AAL2 and ATM Switch Tested

Figure 4.4: A Practical Solution for Creating AAL2 and ATM Switch Environments on the WUGS



Chapter 5

The AAL2 Switch Architecture

5.1 Introduction

This chapter will focus on the software architecture of the AAL2 switch that will run on one or multiple SPCs in the WUGS. First a general system overview is given, with emphasis on the requirements to support a modularised system, and how scheduling of the modules should occur. These modules are then discussed in a particular order, which is influenced by the following factors:

- To degree to which certain units depend upon one another
- The flow of information through the system. As an example, AAL2 packets will first enter the ADU before being passed to the PSU and from there to the ARU and TU mechanisms. These four units may be essentially viewed as an AAL2 pipeline, and are hence discussed in this order.

The chapter concludes with relevant notes on development issues.

5.2 System Overview

The AAL2 switch architecture is modelled as a non-preemptive, cooperative system, with several modules that run independently of one another. As these modules are required to operate in parallel, the resource of the CPU must be shared evenly amongst all processing

entities, and hence context switching is required. The only way to achieve context switching in a native NetBSD environment is to allocate a separate process for each module that is required, and allow the kernel to schedule the individual processes.

There are several drawbacks to this approach. Firstly, for each process that is spawned, a separate memory space must be allocated. There have been attempts to allow processes to share memory, but the mechanisms are complex and non-portable across platforms. Secondly, due to the preemptive nature in which the kernel switches in and out each process, there is no way of knowing when a given code segment will be interrupted. This can result in an unwanted condition in which two processes simultaneously attempt to execute a segment of code that ought to be allocated to only one process at any given time. Critical sections and atomic locks can be used to avoid these race conditions, but these mechanisms add complexity to algorithms and atomic locks must be supported at the hardware level. Finally, the use of processes limits the scalability of a system. As large segments of memory must be shifted with each context switch, the responsiveness of a system with many processes is affected. If a process were to be allocated to each AAL2 path, for example, the AAL2 switch would become sluggish under heavy load.

A better alternative, which alleviates the drawbacks mentioned above, is to make use of threads, which are essentially lightweight processes. The fundamental advantage of threads over processes is that threads are capable of sharing memory amongst themselves, and each thread need only store its own program counter, stack pointer, stack memory and signal table. Each module is therefore implemented as a separate thread, with each thread being scheduled at the process level, and not at the kernel level.

By employing a cooperative strategy amongst the threads in execution, the race conditions discussed above can be eliminated in the following manner. When a thread is picked up by the CPU from a ready or waiting queue, the thread will be given full control to the CPU and must relinquish control at some stage to give other threads a chance to run. This is achieved by an explicit yield instruction from the thread itself, which immediately results in the thread being placed once again on the ready queue. Therefore when data is made available for a particular module or thread, the module will process the data and yield control to the scheduler. Another advantage provided by a cooperative strategy is that routines are not required to be re-entrant, since a cooperative environment will not allow two modules to run the same code segment simultaneously. A re-entrant routine is required to yield the same results when being invoked by multiple threads of execution, and is substantially complex to implement.

It should further be noted that the AAL2 switch is a work conserving system, which implies that the switch is never idle when there is useful work to do. A preemptive system would produce side-effects that would result in a non-work conserving system. A diagram depicting the organisation of the switch architecture is shown in Figure 5.1. Unlike the design diagram of Figure 3.3, more emphasis is placed on the core modules of the switch and their supporting structures. As the Admission Control Unit (ACU) and Interrogator Unit (IU) have not at this stage been implemented, they are either shown as null entities, or have been omitted from the diagram.

The modules that were either partially or completely implemented were:

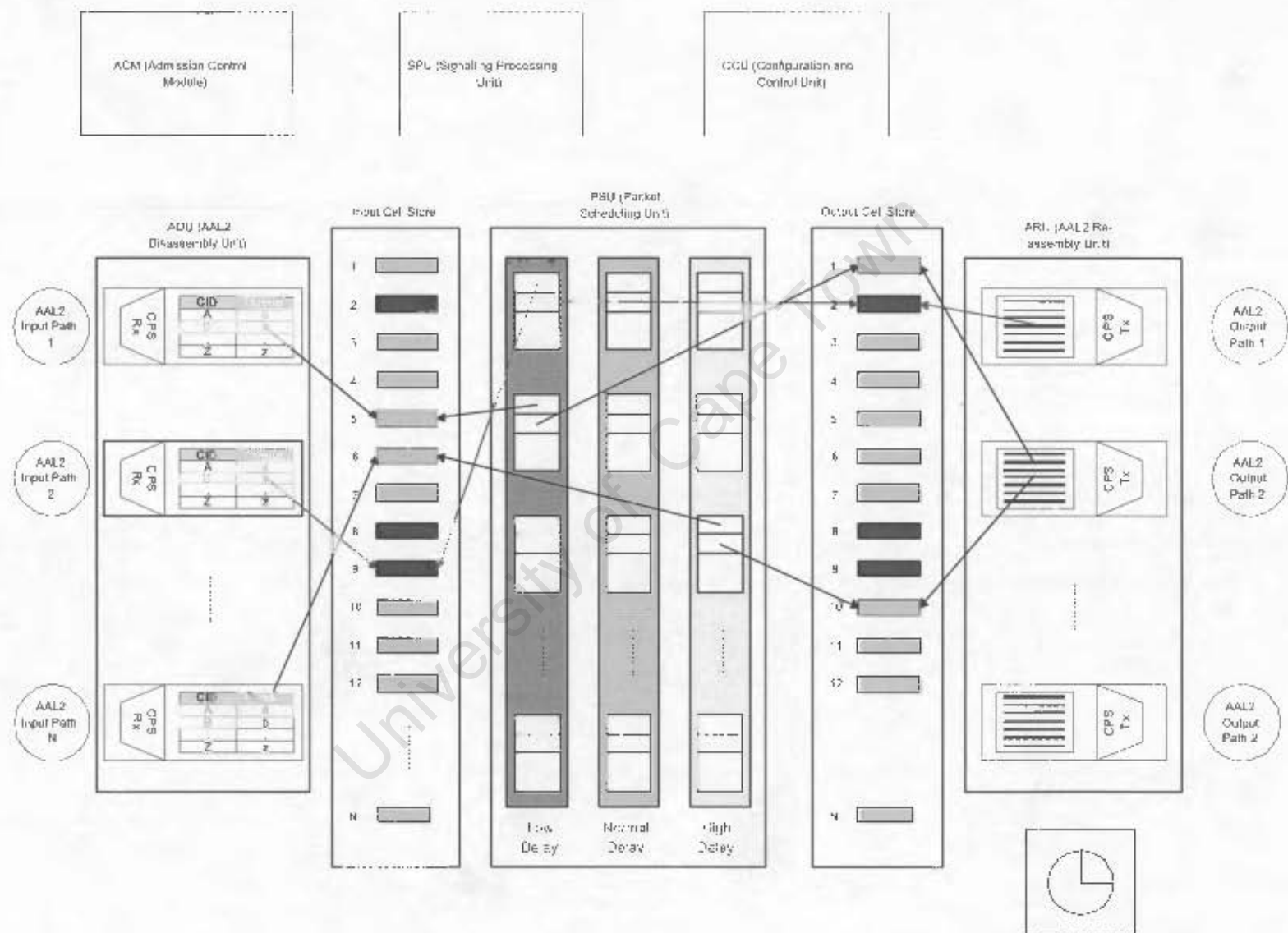
- Signalling Processing Unit (SPU)
- Configuration and Control Unit (CCU)
- AAL2 Disassembly Unit (ADU)
- Packet Scheduling Unit (PSU)
- AAL2 Re-assembly Unit (ARU)
- Timer Unit (TU)

Three important structures that may also be found in the switch, and that are not listed above, are the input packet store, output packet store and CID table. These structures are not considered to be stand-alone modules, but rather as central mechanisms, to which all modules that require them, have equal access. As an example, the Input Packet Store needs to be accessible from both the ADU and PSU, while the Output Packet Store should be accessible from the PSU and ARU, as was pointed out in Chapter 3.

Communication between modules is achieved by the use of events, a standard mechanism used by threads. If a module is waiting for an event, it will be placed on an event queue, and another module will be given a chance to execute. When an event is generated for the module in question, the scheduler will move the module from the event queue to the front of the ready queue, ready for execution. Once the current running thread yields control, the module that had been waiting for the event is finally given a chance to execute. These events are used to emulate several of the interface signals discussed at the end of Chapter 3.

Figure 5.1: Simplified AAL2 Switch Architecture

65



5.3 Operation of the SPU and Processing of Events

The purpose of the SPU, as outlined earlier, is to process incoming AAL2 signalling packets, perform required system updates and generate required signalling packets destined for remote switches. The SPU may be regarded as the unit in the AAL2 switch that most intimately relates to the passing of messages via events, as it is driven by input events before any useful processing may be performed. The SPU can receive any of the following input events:

- **AAL2_IN_SIG_PACKET** specifies that an AAL2 signalling packet has arrived, and is waiting in the Input Packet Store to be processed. A parameter that must be specified is on what queue descriptor the packet is to be found.
- **ADD_AAL2_INPUT_PATH** or **REMOVE_AAL2_INPUT_PATH**, require an input path identifier to be specified. This could take the form of a valid incoming ATM VCI, in which the input AAL2 cells will arrive.
- **ADD_AAL2_OUTPUT_PATH** or **REMOVE_AAL2_OUTPUT_PATH** are identical to their input counterparts, but instead require an outgoing ATM VCI to be specified
- **ADD_CHANNEL_MAPPING** or **REMOVE_CHANNEL_MAPPING** both require a parameter triplet, consisting of the input path created above, a channel ID, from 9 to 255 representing the AAL2 channel that is to be bound or unbound to the outgoing path, respectively, as well as a Type of Service identifier. The reason for not specifying the ITU recommended range from 8 to 255 is due to the fact that channel 8 is reserved for signalling AAL2 channels, and channel 8 is automatically allocated by the SPU for each input path created. If the input path does not exist, the channel ID is not in the specified range, or has already been allocated, and an appropriate response error signal (or event) should be generated.
- **ADD_AAL2_ROUTE** or **REMOVE_AAL2_ROUTE** maps or unlinks an input AAL2 path and associated channel, as bound together above, to an outgoing AAL2 Path, respectively. If the output path does not exist, or a condition occurs regarding the first two parameters as mentioned above, an appropriate error signal should be generated.

At system startup, when the SPU is instantiated, it is not ready to accept input events. It first executes an initialisation routine. Although this routine can be used to create input paths, output paths, channel mappings and AAL2 routes manually, the primary purpose of this initialisation phase is to set up the required mechanisms to process events. In order to be able to receive a set of events, and not just a single one, a structure known as an event ring is required. Each element in the event ring contains a pointer to the following element. The elements in the ring represent all *possible* events that can be generated, and should any of these events be generated, the event(s) will be held in memory until they may be processed. What is then required is for the SPU in its standard operation state, to wait on this event ring and to process the incoming events as they arrive. In the context of the signals discussed above, the input event ring will hold eight possible events, each representing one of the possible input requests.

Not only should a mechanism exist to process and receive events themselves, but also the event parameters. Furthermore the number of these parameters is not fixed. As an example, the `ADD_AAL2_ROUTE` event requires three parameters to be specified, whereas the `ADD_AAL2_INPUT_PATH` specifies only one parameter, and some events may even have no parameters. Therefore each event is bound to a structure known as a message port. A message port represents a queue into which messages representing the parameters are passed. As the message port is bound to a certain event, the appropriate event will automatically be generated as soon as a message is placed on the queue. The unit that is targeted, in this case the SPU, upon receiving the event will know that a message is pending, and will read the message, representing the required parameter off the queue. The question that arises is: If an event requires three parameters to be specified, should three individual messages be queued? The answer to this would be negative, as three individual events would be generated, hence requiring the receiver upon reading the first event to decide on how many additional events are required to fulfil the outstanding information. A far better solution is to pass a pointer as part of the message that is placed on the message port. This pointer refers to a structure that contains the required number of elements. Note that the structure is not passed, only the pointer, hence requiring only a single event to be generated. This process of passing pointers to structures into message ports, which then results in events being generated, is illustrated below in Figure 5.2.

Once the SPU has completed setting up the event ring and associated message ports, it enters the standard operation state, where it will wait for an event to be received. If no events exist, the thread representing the unit is immediately suspended, allowing the

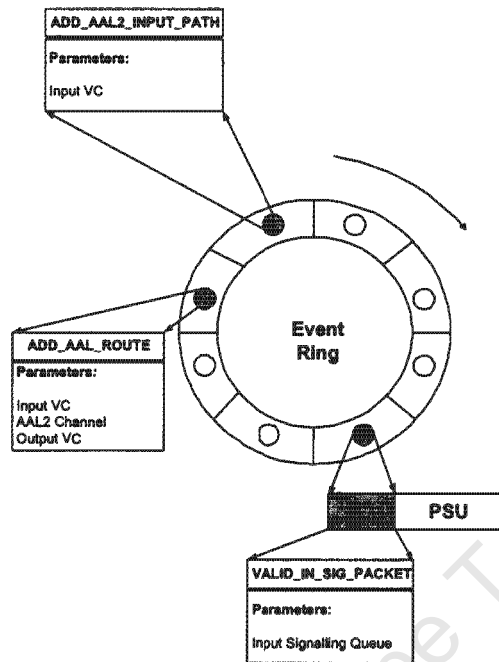


Figure 5.2: Passing Message Ports as Events to the SPU

other threads in the system to run. When an event is received, the event is processed. If updates are required, for example modifying internal tables, these are performed. If an event specifies the existence of a signalling packet, the packet is read off the Input Packet Store and subsequently processed. If output signals are to be generated, appropriate events can be placed in an output event ring, in an identical manner as discussed above. If a target unit is to receive one of these events, it must wait on the same event ring as specified by the sender. Finally, if AAL2 signalling packets are targeted to a remote AAL2 switch, the packet must be placed on the Outgoing Packet Store. The next step would involve determining the output path for which this packet is destined, and notifying the correct ARU instance (that is responsible for generating outgoing AAL2 PDUs for the path in question). The event that would be generated would naturally carry a parameter to inform the ARU where to find the signalling packet in the Output Packet Store.

It should be stressed at this point that by creating the above mechanisms, the SPU is in the state where further work can be carried out in a second research phase, with no prior knowledge required of the internal workings of the AAL2 switch, how the modules (threads) interact with one another, and how AAL2 PDUs are formed into packets and

vice versa. The developer need only concern himself/herself with the relevant interface and processing of these events.

5.4 Operation of the CCU

The CCU forms the core of the AAL2 switch architecture. Although it is responsible for only a small portion of the total processing in the switch, it is the first unit to be given control upon switch startup. The CCU will then instantiate the other modules by starting up their individual threads. When the switch is booted, no parallelism exists and there is only one thread of execution. The CCU performs the following functions to allow the AAL2 switch to reach a state in which incoming information can be processed:

- Create an environment to support threads by loading the relevant libraries
- Initialise a pool of FIFO (First In First Out) queues. When a FIFO queue is requested by a module, a queue is allocated from this pool. There is a reason for initialising this pool up-front. Although at first queues have not been assigned to any use, the memory required for each queue has been requested and is committed as from this instance in time. When at a later stage a queue is required, a queue descriptor need only be assigned to the relevant module instead of the module having to dynamically request memory. Memory requests are expensive and consume processor resources. Hence it is best to perform as much memory allocation as possible in the initialisation stage, before any useful work is required. The amount of memory that is preallocated depends on the maximum number of channels that are to be supported. Therefore, if a good response is required, then a large number of queues should be preallocated. In the event that the pool runs dry, a result of trying to allocate more channels than what internal queues can accommodate, additional memory may then be requested. FIFO queues are required particularly by both Packet Stores, and the queues must be accessible by the ADU and ARU modules, as well as the PSU and SPU.
- Initialise a set of linklists, to be used by the PSU and ARU, the use of which will be explained later.
- Initialise the remaining modules by spawning their threads. These modules are the SPU, PSU and TU. Once these modules have been initialised, they will be given a chance to run once the CCU yields control.

- Enter the steady state. In this case, the CCU yields control. Further functionality that can be included into this module is to receive instructions from a management workstation. The data path in which these instructions would flow is likely to be out of band of the user AAL2 streams, and may even be based on TCP sockets.

5.5 Static (single-instance) and Dynamic (multi-instance) Modules

The modules comprising the AAL2 switch can be placed into two categories. The SPU, CCU, PSU and TU can be referred to as static modules, whereas the ARU and ADU are implemented as dynamic modules. The static modules are created when the switch boots up and remain active throughout the operating lifetime of the AAL2 switch, hence occupying the same memory space. Furthermore there is only one instance of each static module, and each static module need only maintain a single state of global significance in the system.

Dynamic modules are not instantiated at the switch startup cycle, but are created upon demand. In the design chapter, the need was specified to maintain a separate state model for each in-going and outgoing path that exists. As the ADU and ARU represent the processing elements to accommodate these input and output path respectively, it was decided to implement a separate unit instance for each path that is required. For example, if two input paths and three output paths have been specified, there will exist two ADU instances and three ARU instances, respectively.

5.6 Functions and Organisation of the ADU

The ADU is primarily responsible for accepting AAL2 based ATM cells from the IU and extracting the CPS packets from these cells¹. These cells were introduced in the previous chapter and it may once again be stressed that these are in fact 56 byte APIC cells, with a four-byte ATM header (excluding the HEC field) and a four-byte internal APIC header.

For each AAL2 input path that is created, a new ADU instance or thread must be instantiated. Apart from the separate state each ADU must maintain, there was another

¹When the IU is not present, the ADU will accept cells directly from the input link.

important concern that further motivated the breakup of a single ADU into individual instances, which is based on socket behaviour. When an ATM network socket is created, a descriptor (a value used to identify the socket) is returned which can subsequently be used to read and write information to the socket. Furthermore, there are two modes in which a socket can operate, viz. blocking and non-blocking. If a socket is set to the blocking mode, and a read command is encountered for example, the execution of the process or thread will cease until information from the link can be read in. Once a specified number of bytes have been extracted using the socket descriptor, the execution of the process or thread will continue unabated. In the non-blocking mode, if a read command is encountered and no data is available, the process or thread execution will continue uninterrupted, and when information does later arrive, it will automatically be copied to a buffer that was specified to hold the cell contents. Hence in the non-blocking mode, a check must be made on the descriptor to determine whether any information did in fact arrive.

Now if the ADU were a single module, blocking sockets would be relatively simple to implement. However, suppose that the ADU was supporting three simultaneous input paths. Note that for each AAL2 input path, a separate socket descriptor is required. When the ADU attempts to read information from the first socket, representing AAL2 path 1, and no data is available, the ADU will stall until data does arrive on the particular AAL2 path, irrespective of the status of the remaining sockets. If in the event that data never arrived on the first socket, the ADU would remain in this state forever, and the other sockets would not be served from this point forward. Hence such an implementation is clearly unacceptable. Coding this single ADU module using non-blocking sockets also has drawbacks. The code for multiple AAL2 paths would become excessively complex, having to keep state of the degree to which each AAL2 path has been processed. Furthermore, if a large number of AAL2 paths were active at any given time, a check to determine whether any of the sockets representing these paths had valid input data or not would need to be performed on each and every socket. Hence the cost and processing delay would increase linearly with the number of the AAL2 input paths present, even if some or most of these paths are completely idle.

If each ADU instance is coded as a separate thread, it need only concern itself with a single incoming AAL2 path. Therefore if the socket interface is implemented in the blocking mode, the thread will block when no input data for its input path is available. This is perfectly acceptable as other input paths will not be affected. Even if the thread is in the blocking state, and data arrives on another AAL2 path, the thread bound to the AAL2

path will be responsible for collecting the data instead. In this manner, the code for each ADU instance is identical and tremendously simplified. Note that re-entrance is also not a concern, as a different state structure, and hence separate memory area is passed with each ADU that is instantiated. A final concern with blocking sockets is that the standard socket commands supplied with NetBSD block the entire process (consisting of multiple threads) for each blocking socket that does not have data to process. However, an equivalent set of thread-based socket commands only result in the thread being blocked. These commands are based on the same POSIX interface standards as their process-based counterparts.

The unit responsible for creating these ADU instances is the SPU. This will of course occur on a command received as an event, or manually by the SPU itself. Henceforth when reference is made of the ADU, an ADU instance is in fact implied, which in turn corresponds to a unique AAL2 input path. Each ADU holds the following state information:

- The AAL2 path identifier, which represents the ATM VC on which cells arrive for the path.
- The path socket descriptor
- The thread identifier for the ADU. On UNIX-based systems each process is allocated a unique process identifier. In a similar fashion, each thread that is spawned is allocated a unique thread identifier.
- A boolean variable, to indicate whether the path will accept AAL2 signalling requests or not. Note that a path may be allocated by a Management System. Furthermore, for security reasons, it may be a desire to not have a switch accept AAL2 signalling requests from various users.
- A structure used to define the state of the CPS receiver.
- An array of 255 elements, known as the Channel Table, which represents all active AAL2 channels bound to the path. Each entry has an active/non-active variable that is set accordingly. Furthermore each table entry points to both an input queue descriptor and output queue descriptor, which represent indices into the Input Packet Store and Output Packet Store respectively. Note it is not a concern of the ADU to copy packets into the Output Packet Store - this should be performed by the PSU. Furthermore it should be noted that strictly only 248 table entries are required, but the array contains 255 elements to simplify indexing.

Based on this information, when each ADU is first instantiated by the SPU, the following once-off actions are performed:

- Open an APIC connection on the specified VC identifier and bind it to a socket that will be used for future data transfers.
- Allocate a FIFO queue to hold signalling requests. Note that signalling packets are not kept in either of the cell stores, but instead in their own queues.
- Automatically activate channel 8 to accommodate signalling packets, and set the descriptors accordingly.
- Initialise the CPS receiver state machine variables.

Now that the ADU has been initialised it is ready to accept new data. When the ADU receives the 56 byte cell from its associated socket, the first 8 bytes are stripped off and the remaining 48 byte PDU is passed to the ITU-T CPS receiver. Note that this receiver will process the entire PDU before returning control to the calling thread. The CPS receiver was coded directly according to the SDL language specification given in [15], and the interested reader is referred to this specification. Once a complete CPS packet has been formed, the packet is passed as a CPS indication, which means that the packet is being passed to a higher layer with reference to the OSI protocol stack. At this stage, the CID entry in the packet is examined, in which case the packet is treated according to one of the following rules:

- The Channel Table for the input AAL2 path indicates the channel in question is invalid. The obvious cause of this is that the particular channel has not yet been allocated. In this instance, the packet is simply discarded.
- The channel is valid, and a CID of 8 was detected. In this instance, the packet is defined as an AAL2 signalling packet. The packet is "enqueued" onto the signalling queue, and an event is issued to the SPU.
- The channel is valid, and a CID of 9 to 255 was detected. This is the most common scenario. Here the packet is "enqueued" onto the appropriate queue as given by the Channel Table entry for the particular CID. No event need be generated, as the packet will shortly be processed by the PSU.

When control is returned to the main ADU thread from the CPS receiver, the ADU will itself cooperatively yield control to the main system scheduler.

5.7 The PSU and Scheduling

As AAL2 traffic is most likely to be voice traffic with certain real-time expectations, a need for scheduling exists to ensure higher-priority packets endure less delay, and that low priority packets do not starve. The purpose of the PSU is to copy the CPS packets from their input queues to their corresponding output queues, while ensuring packet fairness. Note that the PSU is not concerned with matching packets to output paths - this is done in the last stage by the ARU. If the PSU were to perform a lookup for each packet that was to be copied, the packet would experience an additional delay. It might at this stage seem to the reader that placing the lookup function at the ARU would simply shift this delay further down the AAL2 pipeline, but this is not the case, and will be explained in the section on the ARU.

The PSU defines three scheduling types of service, viz. low delay, normal delay and high delay. Each of these scheduling classes is implemented as a separate linklist consisting of entries known as channel mappings. A channel mapping is simply a structure that contains an input queue descriptor, with a matching output queue descriptor. When the Packet Scheduling Unit (PSU) receives the `ADD_CHANNEL_MAPPING` event, not only does it mark the appropriate Channel Table for the input path active, as indicated earlier, but also informs the PSU that packets will arrive soon, and must be scheduled according to one of the three types of service. This is performed by creating a new channel mapping structure, and initialising the descriptors. Once set, the channel mapping entry is simply appended to either the low delay, normal delay or high delay linklist. It does not make a difference if the channel mapping is appended to the head or tail of the linklist. The only possible side effect is an instantaneous re-shuffling of the order in which the packets in the linklist are processed, but this effect is eliminated on a long term basis².

When the PSU is given control by the system scheduler, it will always process the entries in the low delay linklist first. For each linklist, first the number of elements is obtained. Then each element is traversed and the corresponding packets, if any, are copied from input to

²Longterm here applies to three switching cycles or more. A switching cycle represents the time to cycle through all system modules.

output. Note that as each queue is visited as the linklist is being traversed, more than one packet may be dequeued from the queue. The question arises to whether only one packet should be copied off the queue before visiting the next queue, or whether all available packets ought to be copied off. This, of course, depends on the scheduling scheme. For the low delay scheme, it is more appropriate to copy only one packet off at a time and proceed to the next low delay element. If too many packets are copied off the same queue, the other low delay channels will suffer. Rather sacrifice the quality of a channel on which too many packets are queueing up than jeopardise the quality of all remaining channels. What is done, however, is to keep track of an additional packet flag for the low delay linklist. When an element is dequeued from the queue, a check is also performed to determine whether any remaining elements exist on the queue. If so, the flag is set. If after the first pass through the low delay linklist, the additional packet flag is set, the linklist is simply traversed again. If any queues still have further (3 or more) elements waiting, the flag will be set once again and the linklist will be traversed a third time, and so on. A threshold counter increments for each time the low delay linklist is revisited, and once reached, the low-delay linklist is no longer traversed for the current switch cycle and attention is instead given to the normal delay linklist.

In the second linklist, the linklist is visited only once, but here the number of elements copied off each queue is set to a value known as the Normal Delay Window Size (NDWS). This window size depends on the number of low delay channels as well as the average peak arrival rate for all the low delay channels. If these channels are few or the average peak arrival rate for packets decreases, the NDWS will increase, as there is naturally more time to process normal delay packets. The NDWS will decrease if the opposite behaviour is detected. Only if there is sufficient time remaining to process the high delay channels, taking into account switch overhead as well as the next time channels must be processed, is the high delay linklist traversed. Here all elements are copied from each input queue to output queue, and therefore the linklist need only be traversed once. A scheme that is recommended by the author, but hasn't been implemented yet, is to log the time at which the PSU begins processing the low delay linklist, as well as the completion time of the normal delay linklist. The difference in these times can be used as a measure to determine the time allowed for high delay processing.

Note that CPS packets within the AAL2 switch are always treated as 64 byte structures, even though their size is variable. This is due to the fact that copy mechanisms can operate quickly on fixed sized data, particularly if these mechanisms are ported to hardware. 64-

byte structures are also easily moved across buses, as it is easy to represent 64 bytes as an integral number of bus transfers. Finally, once all elements have been processed, the PSU yields control to the main system scheduler.

5.8 Operation of the ARU

The AAL2 Re-assembly Module is intimately related with the Timer Unit to perform the following primary functions:

- Ensure that AAL2 channels are inserted into the correct outgoing path
- Reassemble CPS packets from the Output Packet Store and output signalling queue into AAL2 PDUs
- Liaise with the TU to ensure that the timing requirements for each outgoing AAL2 path are met

Like the ADU, for each outgoing path that is requested, a separate ARU instance or thread is created by the SPU. Sockets are used to write out the data in a similar way as discussed for the ADU. Hence the same limitations caused by blocking exist and similar action may be used to address such issues. If the APIC link is not ready to accept new data from the ARU, the ARU will suspend execution until new data arrives.

Each ARU holds the following state information:

- The AAL2 output path identifier, which represents the ATM VC on which APIC cells will depart
- The output path socket descriptor
- The thread identifier for the ARU
- The timer-CU limit for the path, specified in time units³.
- A structure to maintain the state of the CPS transmitter.
- A descriptor pointing to the output signalling queue.

³The concept of time units will be discussed in Chapter 6.

- A structure known as a channel set. A channel set is simply a linklist of entries, one of each channel that is to be transmitted on the AAL2 path. Therefore if ten channels are destined for the path, excluding the signalling channel, there will be ten entries in the linklist. Each entry holds a channel ID, and corresponding descriptor that indicates where in the Output Packet Store packets for the channel can be found.

When each ARU is initialised, the following actions are taken:

- Open an APIC connection on the specified VC identifier and bind it to a socket that will be used for future data transfers.
- Allocate a FIFO queue to hold output signalling packets generated from the SPU.
- Initialise the CPS transmitter state machine.

When an AAL2 route is established in the switch, an outgoing AAL2 path must be mapped to a channel and input path. As each input path and channel combination maps directly to an input packet queue, which via the PSU maps directly to an output packet queue, it is simple to obtain the output queue descriptor for a given pointer to an AAL2 input path and corresponding channel. Therefore, the output AAL2 path can “claim” the set of channels that are to be reassembled on the outgoing path. This is simply done by adding an entry to the Channel Set linklist which contains the channel ID and output queue descriptor.

When execution is given to the ARU, the ARU will first check to see whether any signalling packets exist that must pass onto the outgoing link. As signalling packets relate to the control plane, they should be given higher priority over user traffic. Each signalling packet extracted from the output signalling queue will be submitted to the CPS transmitter for processing. When no more signalling packets can be extracted, the ARU can now process the Channel Set. First the number of entries in the Channel Set are obtained. Then the linklist representing this channel set is traversed, and for each queue visited, all packets are extracted and submitted to the CPS transmitter. Note that it is quite in order to extract all packets from each queue, as packets would not be available in the first instance if the PSU had not allocated enough priority to the channel. The primary reason for implementing the Channel Set scheme is to eliminate lookup delays. Only when an AAL2 route is established, is an entry added to the Channel Set. Under low load conditions, there will most likely exist several channels in the Channel Set in which no packets are available.

In this case it may be questionable whether traversing a linklist with some empty elements is really quicker than performing a table lookup. However, under heavy load, far more channels will be active, and traversing the linklist will result in many more hits.

Each packet that is extracted is immediately submitted to the CPS transmitter. The operation of this CPS transmitter was explained in detail in Chapter 2. If the transmitter was in the Idle state, it processes the packet contents, appending them to the newly created AAL2 PDU. If there is still space remaining in the PDU, the transmitter will initialise the timer-CU for the path, by issuing an ADD_TIMER event to the TU, with the timeout value included, specified in milliseconds, as well as the output path that generated the request. With the timer now running, the transmitter enters state Part and returns control to the ARU. As more packets are submitted for processing; provided the PDU does not fill up, the transmitter will always return to state Part. However, two conditions will cause it to change state:

- The CPS transmitter will change to state Full when a packet is delivered by the ARU and the AAL2 PDU is now full and must be sent. In this instance, the PDU is converted to a 56 byte APIC cell. The ATM header is calculated using the Path VC identifier that was specified on creating the path. Note that in the implementation of the AAL2 switch, the ATM VPI is always treated as zero, as the virtual path concept exist at the ATM layer, and is of not of great significance here. The APIC header is calculated to ensure cells enter the WUGS switch and do not simply pass out the link interface again. Finally the APIC cell is submitted to the network layer by issuing a write socket command.

If the PDU is sent off before the timer-CU expires, an event is generated to the TM, known as CANCEL_TIMER, with the output path identifier. As a new PDU will be under construction, the timer-CU can be reset to its default value.

- The CPS transmitter will change to state Send when the appropriate timer-CU for the path has expired. In this instance the TU can directly invoke the CPS transmitter, in the so-called TIMEOUT mode. The conventional mode of calling the transmitter is the CPS_UNITDATA mode, which simply implies more data is being submitted for processing. In the TIMEOUT mode, the partially constructed PDU is padded with zeros, send out on the network link, and the transmitter returns to the IDLE state.

If the transmitter was called in the CPS_UNITDATA mode, it will return control to the ARU, which will itself yield control the main system scheduler. If the transmitter was called in the TIMEOUT mode, it will return control to the TU instead.

5.9 The Timer Unit

Although the TU is itself implemented as a thread, it has the highest priority in the system and is capable of interrupting the entire system process. The principle function of the TU is to coordinate the timers for all active output AAL2 paths, and ensure the relevant transmitter entities are notified on expiry of a timer. The TU can accept two input events:

- ADD_TIMER, with the corresponding output path identifier and timeout value
- CANCEL_TIMER with the output path identifier

Implementing the TU raised several concerns initially. Implementing a single timer on NetBSD is relatively straightforward. A timer structure is created, with the time specified in seconds and microseconds. The system timer is subsequently started, with this structure passed to it. When the timer expires, a system-wide signal known as SIGALRM is generated. To catch this signal, a handler function must have been specified before the timer was invoked. The function of this handler is to notify the ARU that generated the timer in the first instance.

However the system timer provided with NetBSD is unable to accommodate multiple timers. If while the timer is running, and a new time is submitted, the timer simply adjusts to the new time. The question arises as to how can one utilise this single timer in such a way to create the illusion of a multiple timer structure? The answer to this lies in a front-end to the timer function, which is invoked by the TU on receipt of the ADD_TIMER event, simply known as the "add_timer wrapper". As the system timer can only accommodate a single timeout value, the wrapper function must store all incoming timers in such a way so that when the system timer expires, a new timer carefully calculated from the store is submitted to the system. As this wrapper is called with both a timeout value and a path ID, an additional requirement that had to be met was to be able to supply this path ID when the timer expired.

When several timeout values are submitted simultaneously for processing, the logical decision is to always take the most critical timer from this list, i.e. the timer that really ought to expire next. As an example, if two timeouts are requested simultaneously, one to timeout in 3 time units and the second to timeout in 5 time units, it would make sense to choose the first timeout of 3, wait for the timer to expire, and then submit the difference between these two times, i.e. 2 on the second round. What is essentially required is a data structure known as an inverse priority heap. A inverse priority heap is simply a queue that always returns the most critical value, in this case the smallest value upon request. However, what was required was not just simply an inverse priority heap (IPH) that could store scalar values, i.e. timer values, but more complex elements. The time elements are in fact structures that consist of three elements:

- The timeout value indicates when the timer will expire, relative to a reference time (discussed below)
- An array of targets: When the timer expires, all targets in this array and hence all AAL2 output paths specified will be affected. The reason for having an array of targets and not just a single target is simple. If two transmitter state machines happen to specify timeouts that will occur at the same instance in time, this must be accommodated into the timer mechanisms. It does not make sense to issue two simultaneous timeouts, and is also not possible. Rather issue a single timeout and specify that multiple entities should be notified.
- A target count, which simply states how many active targets exist for each timeout value.

When it may be clearly ascertained that no timers indeed are running in the system, a reference time variable is set to zero. The reference time simply keeps track of where on a frame-based time scale timer are interrupting and new timers are added, in order that correct decisions can be made on what values should be passed to the system timer. Apart from the timer elements that can be stored on the heap, there is also a current time element, which has the same timeout and target structure discussed above. The `current_time` element always contains the timeout value, relative to the reference time, when the following timeout will occur. When the `add_timer` function is called for the first time, there will of course be no future timeout value, and hence the current timer structure will be modified accordingly. Note that the heap is only used for the first time when a

timer is already running, and a new timer is specified. If the TU is in this state when the `add_timer` wrapper is called, the following conditions need to be taken into account:

- The new time submitted to the `add_timer` wrapper will expire before the already running timer. In this instance, the current running timer and its matching targets is suspended and placed on the heap. The new time becomes the current time and system timer is invoked once again with this time. When the system timer expires, the original time will be pulled off the heap, the difference in time will be calculated and the difference will be passed to the system timer as the current time.
- The new time submitted is greater than the already running timer. Here, the system timer is left untouched, as it is still considered as the most critical timer in the system. The new time and its targets is simply placed on the priority heap for processing at a later stage. Note that if a timer is placed on the IPH, and another element already exists on the heap of the same value, the target lists are simply merged with one another.
- The new time submitted equals the current time. The system timer is left untouched, but the target list of the current timer is appended to include the new target passed with the new time. Therefore when this timer expires shortly, all affected transmitters will be notified accordingly.

When the current timer expires, the handler function will always be invoked. The handler function first notifies the relevant transmitter entities that PDUs must be sent out, by traversing the target list of the expired timer and generating events to each target (ARU) affected. Then the next timeout value and associated target are extracted from the IPH, if any. If no more timers exists, the TU enters the non-timing stage and the reference time is once again set to zero to start a new reference frame.

Occasionally a timer already set must be cancelled. Timers are cancelled by specifying the target that should not be affected, and not the timeout value. Therefore it is not simple to traverse the IHP to remove the required target, as the IHP uses timer values as comparison elements and not targets. A much simpler scheme is to use a cancelled timer array. When `CANCEL_TIMER` events are received, the timer target is simply added to this array. The timer that was originally specified is still left to expire in the conventional manner. However, in the handler function, the cancelled timer array is examined, and if

the affected targets generated by the expired timers equals an element in the array, the transmitter is no longer notified, creating the illusion of a cancelled timer. Once an element is found in the array, it is simply removed.

5.10 Development Notes

In the preceding sections, it was noted that frequent use is made of FIFO queues, linklists and inverse priority heap structures. For this reason, these structures were coded as generalised structures, and can be extended to any implementation that requires them. These structures and the features that each can provide are discussed in Appendix D.

The programming language selected for the AAL2 switch implementation was C, as C is fast, efficient and constant in the results it produces [26]. Furthermore the NetBSD operating system was implemented using C, as well as the APIC network drivers.

To facilitate the development of the project, sections of the implementation were broken into individual files. Each module is coded as a separate file, as well as the three abstract data types and transmitter and receive state machines. A more detailed description of the code base can be found in Appendix F.

The following external libraries were required to made the development of the AAL2 switch possible:

- The PTH library, or GNU portable threads, required to support modules as threads and inter-module events
- The raw ATM library, required to provide raw ATM socket support, or the creation of "AAL0 cells".
- The generic socket libraries, required for standard socket operation
- The time library, required to implement the process-wide system timer
- The signal library, required to allow a handler function to be attached to the SIGALRM system signal on timer expiry

Chapter 6

Evaluation of the AAL2 Switch

6.1 Introduction

This chapter discusses the evaluation of the AAL2 switch. The first few sections discuss issues relevant to the traffic sources that were used to evaluate the switch, but more details may be found in Appendix C. Subsequent sections discuss tests performed to evaluate operational aspects of the AAL2 switch, such as the ability to switch packets correctly. The remaining sections, and bulk of the chapter, will analyse performance issues relevant to the AAL2 CPS and the effects induced by upper-layer application traffic.

6.2 Source Generator Requirements

In Chapters 1 and 2 AAL2 was defined as an adaptation layer to support the transmission and multiplexing of several variable low-bit rate channels onto an AAL2 Path, or ATM virtual connection. Although the low bit-rate traffic of interest here may be data being submitted to the AAL2 layer, it generally refers to encoded voice traffic generated by applications or middle-ware. Therefore, in order to perform an analysis on the AAL2 capabilities of the system, an application capable of generating these CPS packets is required.

This application or client should fulfil the following requirements:

- The length of the packets generated should not be random, and must be set beforehand, either by means of a script that is parsed on startup, or through a management

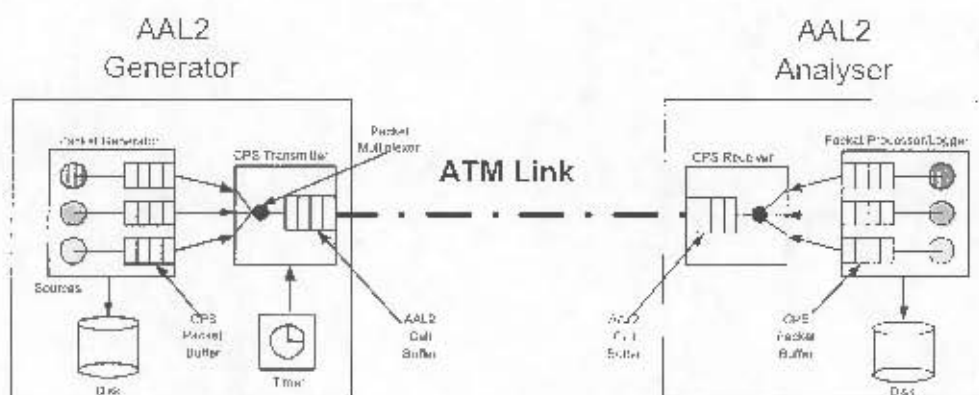


Figure 6.1: Testbed Components to Evaluate the AAL2 Switch

interface. The length of each packet payload may range from 1 byte up to a maximum of 64 bytes. Packets larger than 64 bytes should not be generated. This requirement is set by the ITU-363.2 specification.

- The structure of each CPS packet header should conform to the ITU-363.2 recommendation.
- It should be possible to vary the number of packets that are generated.
- The rate and size of the packets generated should conform to that specified by one of several standardised voice compression codecs that are currently available.
- The contents of the payload of each packet should be randomly generated as if it were valid voice information. This is the simplest method to generate data.
- Each CPS Packet and AAL2 PDU that is generated must be logged to disk for off-line analysis.
- Multiple sources, or multiple users should be supported.

There are many standardised algorithms available for compressing voice and generating the variable-length packets. The algorithms range from fairly simple codecs such as G-711 that perform almost no compression, to highly sophisticated algorithms such as G-729 that produce speech at very low bit rates. The multiplexing feature of AAL2 is best utilised when being applied to lower bit-rate codecs. The codecs of interest are listed in Table

Codec	10 byte packet		20 byte packet		36 byte packet	
	Bandwidth (kbps)	Delay (ms)	Bandwidth (kbps)	Delay (ms)	Bandwidth (kbps)	Delay (ms)
G-726	46.9	2.5	41.5	5		
G-728	23.5	5	20.7	10		
G-729	11.7	10	10.4	20		
GSM-FR					11.9	20

Table 6.1: Standard Compression Codecs

6.1. Although some codecs are described by an exhaustive array of parameters, many of these parameters are not of interest to the network or physical layers. As an example, some parameters attempt to define models that assist in reconstructing speech and comfort noise, both only of relevance to the sending and receiving end application. We need not concern ourselves with these. However, there are two fundamental parameters which may be used to characterise each codec. The source packet distribution refers to the time interval between successive packets. Therefore, if the source packet distribution is 20 ms, then packets from that source will be generated every 20 ms. The second parameter of interest is the packet size of the source. The packet size will vary according to the compression codec that has been selected for evaluation purposes. Both the source packet distribution and packet size for a number of common codecs are shown in the table above.

To model a source, the test client reads data from a configuration file that describes the manner in which packets for the source must be transmitted. These parameters include the codec, packet size, number of packets, and source packet distribution, i.e. the rate at which packets are produced. If multiple users are to be emulated, then multiple sources must be defined in the configuration file. Also defined in the configuration file is the timer-CU that will have global significance to the AAL2 path and hence all channels to be multiplexed onto it. The implementation of the test client and server is discussed in further detail in Appendix C.

6.3 Testbed Components

To perform an evaluation of the AAL2 switch, the following components were required:

- AAL2 Generator, shown in Figure 6.1 to emulate multiple sources and combine their

traffic into a single outgoing AAL2 path.

- AAL2 Analyser, shown in Figure 6.1 to receive AAL2 traffic from the source and reconstruct the individual CPS packets for comparison with those logged at the sending end.
- A WUGS with an AAL2-enabled switch port, i.e. equipped with an SPC.

6.4 System Calibration and Time-scale Normalisation

Figure 6.1 showed how CPS packets could be generated as if they were coming from multiple sources, before being multiplexed onto a single AAL2 connection. To multiplex packets from several sources is simple - just submit the CPS packets when they are generated to the CPS transmitter, and all will be taken care of. If every source that is to be multiplexed generates packets at the same rate, then the CPS transmitter must work at N times the rate at which a single source is transmitting, where N represents the number of sources that are being emulated. For example, as shown in Figure 6.1, three sources are being multiplexed onto a single AAL2 link. If each source generates packets every 30 ms, then the CPS transmitter must be capable of accepting at least three packets within a 30 ms period, i.e. 1 packet every 10 ms. If five sources were being multiplexed instead, then the CPS transmitter would have to accept 5 packets within a 30 ms period, i.e. one 1 packet every 6 ms, regardless of the packet size. Therefore the greater the number of sources in the system, the harder the CPS transmitter must work.

In the above example, it was simple to calculate the rate at which packets would enter the CPS transmitter, since all sources that were defined were characterised by the same codec (packet distribution and packet size). However, assume now that the three sources, called A, B, and C generate packets every 5 ms, 10 ms and 20 ms respectively, as shown in Figure 6.2. Exactly how fast must the CPS transmitter be able to accept CPS packets? By examining the figure, one can see that as 20 ms is the slowest rate, it defines the period or cycle over which the same behaviour will occur again and again. In other words, the distribution of packets is the same in the first 20 ms period as well as all subsequent periods of the same size. Therefore one need only analyse the system over a single 20 ms interval, much in the same way as performing an analysis on a rate-monotonic system. In 20 ms, seven packet are generated. Therefore the CPS transmitter must be capable of accepting a

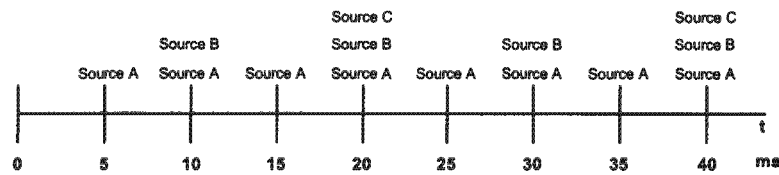


Figure 6.2: Multiplexing Three Sources onto an AAL2 Link

new packet approximately every $20/7$, or 3 ms. Since 3 ms is less than the smallest source packet rate or distribution of 5 ms (corresponding to Source A), one can conclude that a CPS transmitter that can accept packets every 3 ms would be sufficient.

In the AAL2 generator, an interrupt will trigger the transmitter whenever a packet is ready. Therefore, for the example above, the interrupt mechanism must be fast enough to trigger the transmitter approximately every 3 ms or more. If the system can only handle interrupts with a time granularity of 10 ms or more, then the generator will not be able to transmit packets fast enough. This was found to be an issue with the NetBSD operating system, which can only generate interrupts every 10 ms.

The only solution to such a problem is to slow down the rate at which packets are generated, in order that the operating system can keep up. To do this, the smallest period at which timers can operate, i.e. 10 ms, is declared to be 1 base unit. 1 base unit will then correspond to a time period of 1 ms. Therefore a real time-interval of 3 ms will be represented in the generator as 3 base units, or 30 ms. In this way, a thorough performance investigation can still be conducted as the results can simply be scaled back to real time-units if required.

To ensure that for the base time selected, sufficient processing capability was available, a sequence of packets was generated every 10 base time units. Each packet generated was logged, as well as the time at which the packet was generated. The log files were analysed upon completion of the run, and it was found that the time increments were indeed constant. If the rate at which the packet times increased deviated slightly (upward) from the norm, the base time would have been too small.

6.5 End-to-end Operational Evaluation of the AAL2 CPS

To analyse the correct operation of both transmitter and receiver, the following requirements had to be met:

```

#Packet 0, CID=17, LI=14
17 56 0 245 26 157 69 39 186 226 168 59 130 156 172 123 101 132
#Packet 1, CID=17, LI=14
17 56 0 109 85 81 147 139 90 29 22 67 72 188 250 106 225 208
#Packet 2, CID=17, LI=14
17 56 0 247 10 13 249 165 200 239 234 37 228 126 180 100 236 67
#Packet 3, CID=17, LI=14
17 56 0 62 152 249 180 103 21 149 4 200 254 18 172 142 13 131
#Packet 4, CID=17, LI=14
17 56 0 160 93 215 21 104 196 226 237 21 42 236 107 85 180 156
#Packet 5, CID=17, LI=14
17 56 0 206 138 30 47 59 42 244 52 18 92 51 240 138 17 186
#Packet 6, CID=17, LI=14
17 56 0 138 164 124 145 130 59 66 114 172 101 51 151 41 232 238
#Packet 7, CID=17, LI=14
17 56 0 58 128 238 66 196 126 172 156 48 210 76 109 122 41 183
#Packet 8, CID=17, LI=14
17 56 0 208 180 186 135 84 166 248 212 43 118 109 20 189 76 97
#Packet 9, CID=17, LI=14
17 56 0 196 209 115 100 32 54 230 83 130 71 198 231 199 21 177
#Packet 10, CID=17, LI=14
17 56 0 121 90 158 103 251 28 220 208 212 240 13 185 171 25 2
#Packet 11, CID=17, LI=14
17 56 0 89 180 29 220 9 176 113 9 51 19 167 192 222 132 62

```

Figure 6.3: CPS Packet Logfile showing 12 Packets, each of length 15 and CID of 17

- Every packet received at the analyser would have to match the corresponding source packet exactly.
- Packets would have to arrive in the correct order in which they were sent.
- At the transmitter, any PDUs that were sent out due to the timer expiring, and not because they were full, would have to be checked, to ensure that the correct amount of padding had been inserted.

A very simple method for comparing streams of packets to determine their accuracy, was to log all packets that were generated at the source end, as well as all packets received at the remote end. A typical logfile of packets is shown in Figure 6.3.

Once both the sending and receiving end log files were extracted, the UNIX utility “diff” was applied to compare the two files and find the difference, hence indicating any non-matching lines that were found. In the earlier development stages, this utility helped tremendously in diagnosing faults both at the transmitter and the receiver. A stage has been reached where end-to-end AAL2 transmission can take place indefinitely, without any errors occurring. Hence, when the files were analysed off-line, their contents were identical, implying packets had arrived uncorrupted and in the correct sequence. Occasionally, the source log file would have one or two additional packets than the receiver log file. This was

```

#Packet 0, CID=20, LI=9
20 36 0 253 116 91 14 205 221 153 51 161 114
#Packet 1, CID=20, LI=9
20 36 0 223 92 113 167 2 75 220 182 36 116
#Packet 2, CID=20, LI=9
20 36 0 234 142 194 243 254 129 170 90 222 186
#Packet 3, CID=20, LI=9
20 36 0 123 5 27 232 209 56 181 23 161 182
#Packet 4, CID=20, LI=9
20 36 0 94 49 230 12 28 31 66 67 190 4
#Packet 5, CID=20, LI=9
20 36 0 143 62 232 128 28 104 20 133 143 5
#Packet 6, CID=20, LI=9
20 36 0 94 162 24 191 131 76 251 191 204 17
#Packet 7, CID=20, LI=9
20 36 0 38 3 89 57 191 29 150 191 120 44
#Packet 8, CID=20, LI=9
20 36 0 57 44 137 44 148 50 27 109 174 158
#Packet 9, CID=20, LI=9
20 36 0 20 78 213 131 245 23 37 23 214 4
#Packet 10, CID=20, LI=9
20 36 0 195 74 31 86 56 128 11 152 168 102
#Packet 11, CID=20, LI=9
20 36 0 158 204 100 100 25 185 139 160 129 26

```

Figure 6.4: CPS Packet Logfile showing 12 Packets, each of length 10 and CID of 20

caused by the receiver having not received the final PDU (as it had not been constructed), hence not being able to construct the final packets.

The logfiles for various tests that were performed cannot be shown here, due to their size. However, tests were performed for packet sizes of up to 64 octets, where the number of packets in each case was increased to 10000 and transmitted in each case. In all the tests performed, both logfiles were found to match one another. To allow for more processing capability in the AAL2 generator, all packets from a particular source file were first read into memory before being transmitted. The generator can write out packets at a much faster rate if they are sourced from memory. By continually having to read each packet from a file, a lot of time and processing power would be wasted, which could have been used elsewhere (for example, by supporting more AAL2 users).

To analyse the effect of the timer-CU at the transmitting end, outgoing PDUs were logged and their contents were analysed. The figures above and below show the process of performing such an analysis. First, a series of packets transmitted every 10 time units, each of length 10 bytes is shown in Figure 6.4. The outgoing PDUs formed from these packets were also logged.

To analyse what the effect would be of varying the timer-CU on the contents of outgoing PDUs, three tests were performed.

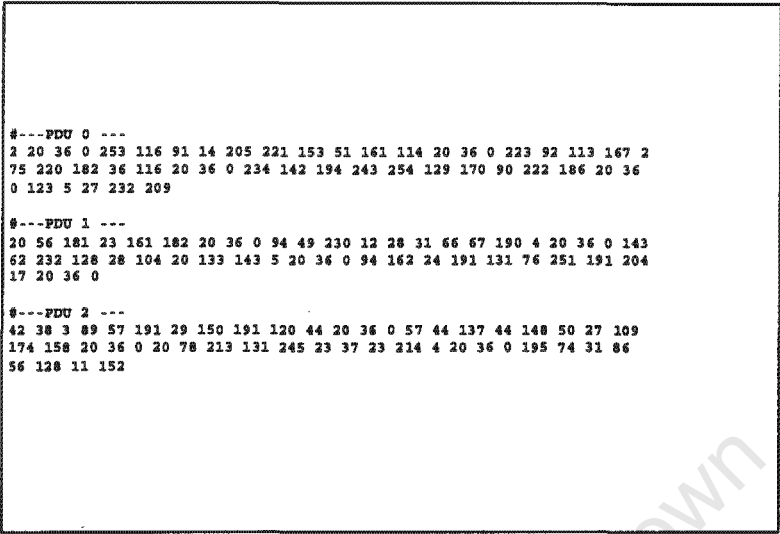


Figure 6.5: PDU Contents for Large Timer-CU value

6.5.1 Maximum Efficiency Test

For the first experiment, the timer-CU was set to a relatively high value of 50 time units. The result of this was simply that the timer had no effect on the outgoing traffic whatsoever, and PDUs were completely filled with packet data. This is shown clearly in the truncated logfile of Figure 6.5.

Another effect that was noted was that half the contents of the second-last packet, as well as the last packet itself, did not appear in the logfile of the figure above. This was simply due to the fact that the transmitter had been waiting indefinitely for additional packet data to arrive to fill the last PDU.

6.5.2 Standard Efficiency Test

The same test was repeated with a timer-CU set to 20 time units. The logfile of this test is shown in Figure 6.6.

In this case a distinct padding of zeros occurred at the end of each transmitted PDU. As soon as two packets had been submitted, the timer expired and the PDU was sent, as shown above. It was simple to note how the efficiency dropped substantially with the lower timer value, as almost half of each PDU remained under-utilised. Note how the remaining

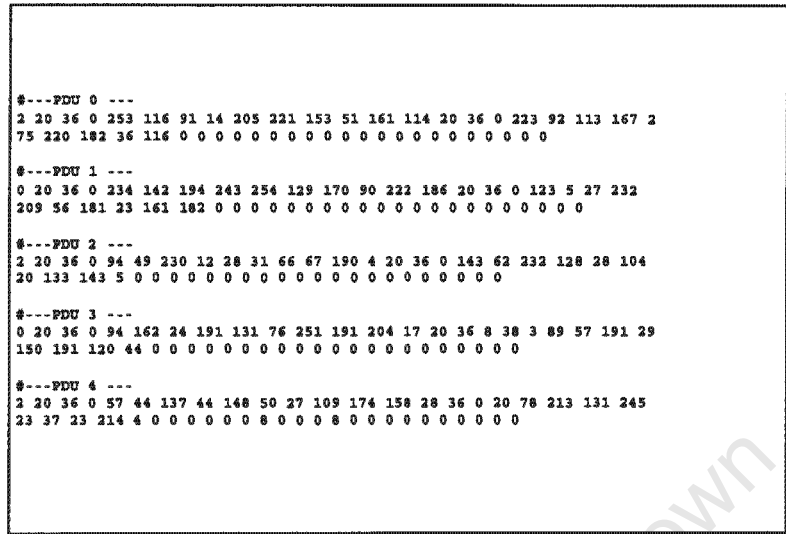


Figure 6.6: PDU Contents for Timer-CU of 20

two packets were also not present, due to the client terminating after only 12 packets were generated.

6.5.3 Low Efficiency Test

The final test was performed by setting the timer-CU to its lowest possible value of 10¹ time units, as shown in Figure 6.7. It can be seen in the corresponding logfile how each PDU carried only a single CPS packet.

6.6 AAL2 Switch Evaluation

The section will evaluate the ability of the AAL2 switch to switch at the packet level, by monitoring CPS packets for each AAL2 path on the output side of the switch. To determine whether the correct operations were being achieved, four tests were conducted.

¹A timer-CU less than the packet inter-arrival time is non-causal.

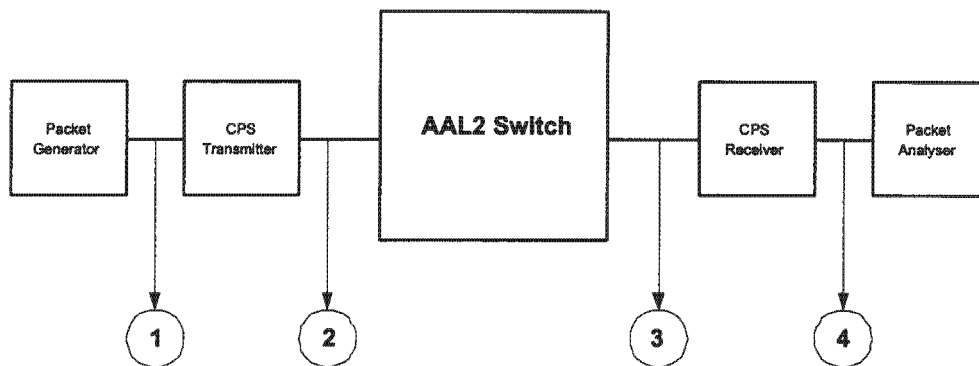


Figure 6.8: Analysing Data at Four Points

By probing all four points as mentioned above, instead of just the end-points, the process of troubleshooting faults on the AAL2 path was simplified. As an example, had the logfiles containing 48-byte PDUs matched, but not the CPS-packet-based logfiles, then the location of the fault would not have been in the AAL2 switch itself, but instead at either of the two end-stations.

10 000 CPS packets of varying attributes were produced, and both sets of logfiles were compared. In all cases, no corruption of the data, at both the packet level and AAL2 level, was detected. Furthermore, no packets were mis-inserted, or appeared at the wrong outputs, which was the desired result for this experiment. All packets and AAL2 PDUs were also found to arrive in the order in which they were sent. The logfiles produced have not been included, due to their size, but do resemble those of Figure 6.4 and Figure 6.5.

6.6.2 AAL2 Path Compression Test

In this test, the effect of modifying internal routes in the switch and the corresponding result on the output traffic (logs) was analysed. The switch was configured in the following way:

- The AAL2 packet generator was configured to produce AAL2 traffic from two sources onto a single outgoing AAL2 path. The logfiles for both these sources, as well as the traffic for the AAL2 path is shown in Figure 6.9. Source A is shown in green on the figure, where each packet is 10 bytes long and is bound to AAL2 channel 12. The second source, Source B, is shown in blue and is bound to channel 24. On the

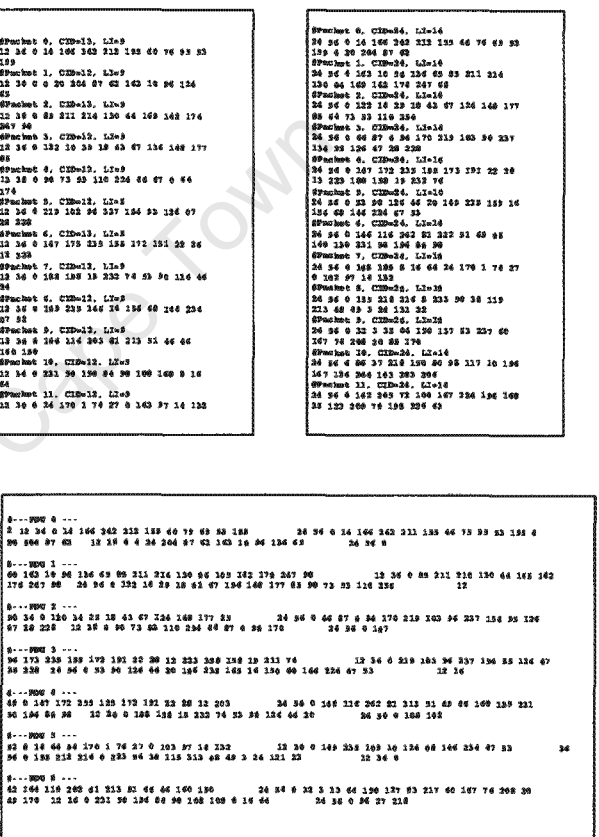


Figure 6.9: CPS Packet and AAL2 Traffic Logs from 2 Sources

corresponding outgoing PDUs, one can see how the CPS packets are interleaved with one another, by simply noting the alternating colours.

- To allow the switch to accept this AAL2 traffic, a single ADU was initialised with the same AAL2 path ID as specified by the AAL2 generator².
- Two channel mappings were created in the switch to bind both the above-mentioned channels to this single input path.
- Finally two ARUs were initialised to allow CPS packets to be encapsulated onto two possible outgoing paths. The first path was configured with an AAL2 Path ID of 100, and the second AAL2 path was configured with an ID of 101. We shall refer to these output AAL2 paths as output path A and output path B, respectively.

²Note that traffic from an AAL2 generator passes directly to the input port of the WTUGS, where the AAL2 switch module can be found, and does not undergo any VP/VC switching.

Three small experiments were performed. In the first experiment, the result of which is shown in the first column of Figure 6.10, the channel set of output path A was configured to include all packets on channel 12, whereas the channel set for the second path (path B) was set to include traffic on channel 24. Since a single input AAL2 path exists, only the channel ID and output path need to be specified when creating an AAL2 route through the switch, as the input path would always have the same value. The result achieved is relatively straightforward - all traffic shown in the logs for output path A only included that obtained from source A, whereas the traffic found on output path B was exclusively that of source B.

In the second experiment, shown in the second column of Figure 6.10, the channels bound to both output paths were simply swapped, so that output path A now contained the traffic from source B, and vice versa.

The third experiment, shown in the third column of Figure 6.10, showed how the AAL2 switch could be used to concentrate traffic onto a single output path. Here the channel set for output path A was set to include traffic from both sources, whereas no channel set for output path B was defined. The result was that all traffic was now concentrated onto the single path. Had one of the channels not been specified in this path's channel set, packets belonging to the channel would not have been able to pass through the switch, and would simply have been discarded.

6.6.3 AAL2 Path Expansion Test

In the previous test, only one input AAL2 path was assumed. However, it would be more practical to evaluate the ability of the switch to accept data on multiple input paths. Figure 6.11 shows three sources, sources A, B and C, shown in red, green and blue, respectively. The components were configured on the following way:

- Sources A, B and C were bound to channels 20, 30 and 40 respectively, with packet lengths of 10 bytes, 10 bytes and 12 bytes respectively.
- Source A was transmitted on input Path A, and sources B and C were multiplexed onto input path B.
- Therefore 2 ADUs were initialised to accept traffic on two independent paths. For each ADU, the corresponding channel mappings were added.

Figure 6.10: Routing AAL2 Traffic to Different Outputs

```
#---PDU 0 on VC 101---
2 24 56 0 14 166 242 212 155 46 79 99
53 155 4 20 204 87 62 24 56 0 163 10
96 136 65 85 211 214 130 44 169 142
174 247 98 24 56 0 122 14 29 18 43 67
136 148

#---PDU 0 on VC 100---
2 12 36 0 14 166 242 212 155 46 79 99
53 155 12 36 0 4 20 204 87 62 163 10
96 136 65 12 36 0 85 211 214 130 44
169 142 174 247 98 12 36 0 122 14 29
18 43

#---PDU 1 on VC 101---
28 177 85 90 73 53 110 254 24 56 0 46
87 4 94 170 219 103 96 237 154 95 126
47 28 228 24 56 0 147 172 235 155 172
191 22 28 12 223 188 158 19 232 74 24
56 0 53

#---PDU 1 on VC 100---
20 67 136 148 177 85 12 36 0 90 73 53
110 254 46 87 4 94 170 12 36 0 219 103
96 237 154 95 126 47 28 228 12 36 0
147 172 235 155 172 191 22 28 12 223
12 36 0

#---PDU 2 on VC 101---
58 90 126 46 20 149 235 169 16 156 68
146 234 47 53 24 56 0 146 116 202 81
213 51 49 46 160 150 231 50 196 84 98
24 56 0 108 108 8 16 64 54 170 1 76 27
0 103

#---PDU 2 on VC 100---
42 188 158 19 232 74 53 90 126 46 20
12 36 0 149 235 169 16 156 68 146 234
47 53 12 36 0 146 116 202 81 213 51 49
46 160 150 12 36 0 231 50 196 84 98
108 108 8
```

```
#---PDU 0 on VC 100---
2 24 56 0 14 166 242 212 155 46 79 99
53 155 4 20 204 87 62 24 56 0 163 10
96 136 65 85 211 214 130 44 169 142
174 247 98 24 56 0 122 14 29 18 43 67
136 148

#---PDU 0 on VC 101---
2 12 36 0 14 166 242 212 155 46 79 99
53 155 12 36 0 4 20 204 87 62 163 10
96 136 65 12 36 0 85 211 214 130 44
169 142 174 247 98 12 36 0 122 14 29
18 43

#---PDU 1 on VC 100---
28 177 85 90 73 53 110 254 24 56 0 46
87 4 94 170 219 103 96 237 154 95 126
47 28 228 24 56 0 147 172 235 155 172
191 22 28 12 223 188 158 19 232 74 24
56 0 53

#---PDU 1 on VC 101---
20 67 136 148 177 85 12 36 0 90 73 53
110 254 46 87 4 94 170 12 36 0 219 103
96 237 154 95 126 47 28 228 12 36 0
147 172 235 155 172 191 22 28 12 223
12 36 0

#---PDU 2 on VC 100---
58 90 126 46 20 149 235 169 16 156 68
146 234 47 53 24 56 0 146 116 202 81
213 51 49 46 160 150 231 50 196 84 98
24 56 0 108 108 8 16 64 54 170 1 76 27
0 103

#---PDU 2 on VC 101---
42 188 158 19 232 74 53 90 126 46 20
12 36 0 149 235 169 16 156 68 146 234
47 53 12 36 0 146 116 202 81 213 51 49
46 160 150 12 36 0 231 50 196 84 98
108 108 8
```

```
#---PDU 0 on VC 100---
2 24 56 0 14 166 242 212 155 46 79 99
53 155 4 20 204 87 62 12 36 0 14 166
242 212 155 46 79 99 53 155 12 36 0 4
20 204 87 62 163 10 96 136 65 24 56 0

#---PDU 1 on VC 100---
60 163 10 96 136 65 85 211 214 130 44
169 142 174 247 98 24 56 0 122 14 29
18 43 67 136 148 177 85 90 73 53 110
254 12 36 0 85 211 214 130 44 169 142
174 247 98 24

#---PDU 2 on VC 100---
70 56 0 46 87 4 94 170 219 103 96 237
154 95 126 47 28 228 12 36 0 122 14 29
18 43 67 136 148 177 85 12 36 0 90 73
53 110 254 46 87 4 94 170 24 56 0 147

#---PDU 3 on VC 100---
56 172 235 155 172 191 22 28 12 223
188 158 19 232 74 24 56 0 53 90 126 46
20 149 235 169 16 156 68 146 234 47 53
12 36 0 219 103 96 237 134 95 126 47
28 228 24 56

#---PDU 4 on VC 100---
66 0 146 116 202 81 213 51 49 46 160
150 231 50 196 84 98 12 36 0 147 172
235 155 172 191 22 28 12 223 12 36 0
188 158 19 232 74 53 90 126 46 20 24
56 0 108 108

#---PDU 5 on VC 100---
52 8 16 64 54 170 1 76 27 0 103 97 14
132 24 56 0 158 218 216 8 233 50 38
119 213 48 49 3 24 131 22 12 36 0 149
235 169 16 156 68 146 234 47 53 24 56
0
```

- A single ARU was initialised. A channel set was initialised to include all three channels, hence defining three separate AAL2 routes through the switch:
 - AAL2 Route 1 carried data from input path A on channel 20 to the output path.
 - AAL2 Route 2 carried data from input path B on channel 30 to the output path.
 - AAL2 Route 3 carried data from input path B on channel 40 to the output path.

The result of this experiment is shown in Figure 6.12. One would expect the traffic from both paths and all three channels to be concentrated onto the single output path, and as shown, this is indeed the case. CPS packets from all three sources are interleaved with one another, and the colours ought to assure the reader that no packets are missing, out of order or corrupted.

6.6.4 Signalling Packet Receive Test

In order to monitor the flow of signalling packets in the switch, a source was specified in the standard manner as was performed for the tests above. However, in this case, the CID entry of several packets at random positions in the file was changed to 8, thus creating fictitious signalling packets. These packets were streamed into the AAL2 switch, and the effects noted. For each packet received, the ADU would queue packets onto the relevant queues. Once the CPS receiver has completed processing the current PDU, an event was generated to the SPU. Included with the event was a message with the number of packets, the queue where the packets could be found as well the AAL2 path on which the packets occurred. In all cases, the signalling packets were retrieved by the SPU for processing.

Figure 6.11: Traffic Logs for 3 Sources

<p>#Packet 0, CID=20, LI=9 20 36 0 208 152 54 17 180 92 15 99 231 72 #Packet 1, CID=20, LI=9 20 36 0 179 20 127 140 0 251 4 28 119 230 #Packet 2, CID=20, LI=9 20 36 0 24 96 240 165 107 11 60 130 54 232 #Packet 3, CID=20, LI=9 20 36 0 188 167 227 14 100 154 67 39 13 104 #Packet 4, CID=20, LI=9 20 36 0 208 87 179 132 108 112 119 15 47 226 #Packet 5, CID=20, LI=9 20 36 0 229 131 139 54 136 118 124 107 36 115 #Packet 6, CID=20, LI=9 20 36 0 251 31 173 143 110 185 94 213 70 147 #Packet 7, CID=20, LI=9 20 36 0 245 22 147 92 128 58 177 155 190 176 #Packet 8, CID=20, LI=9 20 36 0 212 180 136 11 174 35 55 102 210 113 #Packet 9, CID=20, LI=9 20 36 0 128 181 44 156 200 129 35 90 83 59 #Packet 10, CID=20, LI=9 20 36 0 192 4 93 192 115 137 133 20 191 68 #Packet 11, CID=20, LI=9 20 36 0 137 250 132 249 200 129 31 244 139 22</p>	<p>#Packet 0, CID=30, LI=9 30 36 0 208 152 54 17 180 92 15 99 231 72 #Packet 1, CID=30, LI=9 30 36 0 179 20 127 140 0 251 4 28 119 230 #Packet 2, CID=30, LI=9 30 36 0 24 96 240 165 107 11 60 130 54 232 #Packet 3, CID=30, LI=9 30 36 0 188 167 227 14 100 154 67 39 13 104 #Packet 4, CID=30, LI=9 30 36 0 208 87 179 132 108 112 119 15 47 226 #Packet 5, CID=30, LI=9 30 36 0 229 131 139 54 136 118 124 107 36 115 #Packet 6, CID=30, LI=9 30 36 0 251 31 173 143 110 185 94 213 70 147 #Packet 7, CID=30, LI=9 30 36 0 245 22 147 92 128 58 177 155 190 176 #Packet 8, CID=30, LI=9 30 36 0 212 180 136 11 174 35 55 102 210 113 #Packet 9, CID=30, LI=9 30 36 0 128 181 44 156 200 129 35 90 83 59 #Packet 10, CID=30, LI=9 30 36 0 192 4 93 192 115 137 133 20 191 68 #Packet 11, CID=30, LI=9 30 36 0 137 250 132 249 200 129 31 244 139 22</p>	<p>#Packet 0, CID=40, LI=11 40 44 0 208 152 54 17 180 92 15 99 231 72 179 20 #Packet 1, CID=40, LI=11 40 44 0 127 140 0 251 4 28 119 230 24 96 240 165 #Packet 2, CID=40, LI=11 40 44 0 107 11 60 130 54 232 188 167 227 14 100 154 #Packet 3, CID=40, LI=11 40 44 0 67 39 13 104 208 87 179 132 108 112 119 15 #Packet 4, CID=40, LI=11 40 44 0 47 226 229 131 139 54 136 118 124 107 36 115 #Packet 5, CID=40, LI=11 40 44 0 251 31 173 143 110 185 94 213 70 147 245 22 #Packet 6, CID=40, LI=11 40 44 0 147 92 128 58 177 155 190 176 212 180 136 11 #Packet 7, CID=40, LI=11 40 44 0 174 35 55 102 210 113 128 181 44 156 200 129 #Packet 8, CID=40, LI=11 40 44 0 35 90 83 59 192 4 93 192 115 137 133 20 #Packet 9, CID=40, LI=11 40 44 0 191 68 137 250 132 249 200 129 31 244 139 22 #Packet 10, CID=40, LI=11 40 44 0 12 122 119 53 82 130 61 97 111 5 231 91 #Packet 11, CID=40, LI=11 40 44 0 237 175 209 82 186 108 166 9 63 118 129 159</p>
---	---	---

Figure 6.12: Concentrating 3 AAL2 channels on 2 AAL2 paths onto a single Output Path

```

8---PDU 0 ---
2 20 36 0 208 152 54 17 180 92 15 99
231 72 20 36 0 179 20 127 140 0 251
4 28 119 230 20 36 0 24 96 240 165
107 11 60 130 54 232 20 36 0 188 167
227 14 100

8---PDU 1 ---
20 154 67 39 13 104 20 36 0 208 87
179 132 108 112 119 15 47 226 20 36
0 229 131 139 54 136 118 124 107 36
115 20 36 0 251 31 173 143 110 185
94 213 70 147 20 36 0

8---PDU 2 ---
42 245 22 147 92 128 58 177 155 190
178 20 36 0 212 180 136 11 174 35 55
102 210 113 20 36 0 128 181 44 156
200 129 35 90 83 59 20 36 0 192 4 93
192 115 137 133 20

```

```

8---PDU 0 ---
2 30 36 0 208 152 54 17 180 92 15 99
231 72 40 44 0 208 152 54 17 180 92
15 99 231 72 179 20 30 36 0 179 20
127 140 0 251 4 28 119 230 40 44 0
127 140 0

8---PDU 1 ---
36 251 4 28 119 230 24 96 240 165 30
36 0 24 96 240 165 107 11 60 130 54
232 40 44 0 107 11 60 130 54 232 188
167 227 14 100 154 30 36 0 188 167
227 14 100 154 67

8---PDU 2 ---
14 39 13 104 40 44 0 67 39 13 104
208 87 179 132 108 112 119 1 5 30 36
0 208 87 179 132 108 112 119 15 47
226 40 44 0 47 226 229 131 139 54
136 118 124 107 36 115 30

8---PDU 3 ---
48 36 0 229 131 139 54 136 118 124
107 36 115 40 44 0 251 31 173 143
110 185 94 213 70 147 245 22 30 36 0
251 31 173 143 110 185 94 213 70 147
40 44 0 147 92 128 58

8---PDU 4 ---
34 177 155 190 176 212 180 136 11 30
36 0 245 22 147 92 128 58 177 155
190 176 40 44 0 174 35 55 102 210
113 128 181 44 156 200 129 30 36 0
212 180 136 11 174 35 55 102

8---PDU 5 ---
9 210 113 40 44 0 35 90 83 59 192 4
93 192 115 137 133 20 30 36 0 128
181 44 156 200 129 35 90 83 59 40 44
0 191 68 137 250 132 249 200 129 31
244 139 22 30 36

8---PDU 6 ---
46 0 192 4 93 192 115 137 133 20 191
68 40 44 0 12 122 119 53 82 130 61
97 111 5 231 92 30 36 0 137 250 132
249 200 129 31 244 139 22 40 44 0
237 175 209 82 106

```

```

8---PDU 0 ---
2 40 44 0 208 152 54 17 180 92 15 99 231 72
179 20 40 44 0 127 140 0 251 4 28 119 230 24
96 240 165 40 44 0 107 11 60 130 54 232 188
167 227 14 100 154 20 36

8---PDU 1 ---
44 0 208 152 54 17 180 92 15 99 231 72 20 36
0 179 20 127 140 0 251 4 28 119 230 20 36 0
24 96 240 165 107 11 60 130 54 232 40 44 0
67 39 13 104 208 87 179

8---PDU 0 ---
2 30 36 0 208 152 54 17 180 92 15 99 231 72
30 36 0 179 20 127 140 0 251 4 28 119 230 30
36 0 24 96 240 165 107 11 60 130 54 232 30
36 0 188 167 227 14 100

8---PDU 2 ---
22 132 108 112 119 15 40 44 0 47 226 229 131
139 54 136 118 124 107 36 115 40 44 0 251 31
173 143 110 185 94 213 70 147 245 22 20 36 0
188 167 227 14 100 154 67 39 13

8---PDU 3 ---
4 104 20 36 0 208 87 179 132 108 112 119 15
47 226 20 36 0 229 131 139 54 136 118 124
107 36 115 20 36 0 251 31 173 143 110 185 94
213 70 147 40 44 0 147 92 128 58

8---PDU 1 ---
20 154 67 39 13 104 30 36 0 208 87 179 132
108 112 119 15 47 226 30 36 0 229 131 139 54
136 118 124 107 36 115 30 36 0 251 31 173
143 110 185 94 213 70 147 30 36 0

8---PDU 4 ---
34 177 155 190 176 212 180 136 11 40 44 0
174 35 55 102 210 113 128 181 44 156 200 129
40 44 0 35 90 83 59 192 4 93 192 115 137 133
20 40 44 0 191 68 137 250 132 249

8---PDU 5 ---
24 200 129 31 244 139 22 20 36 0 245 22 147
92 128 58 177 155 190 176 20 36 0 212 180
136 11 174 35 55 102 210 113 20 36 0 128 181
44 156 200 129 35 90 83 59 40 44

8---PDU 2 ---
42 245 22 147 92 128 58 177 155 190 176 30
36 0 212 180 136 11 174 35 55 102 210 113 30
36 0 128 181 44 156 200 129 35 90 83 59 30
36 0 192 4 93 192 115 137 133 20

```

6.7 AAL2 Performance Metrics

Having evaluated the functional operation of the AAL2 switch, the following sections shall concentrate on performance issues, particularly that of the AAL2 transmitter and its timer mechanism. The following performance parameters are of interest:

- **PDU Efficiency**

The PDU efficiency is a measure of how well a given AAL2 PDU has been utilised. As a goal of the AAL2 protocol is to improve the statistical multiplexing gain of ATM for voice, payload efficiency is naturally of concern.

- **Timer-CU**

Although the purpose of the timer-CU is to ensure that delay requirements of voice are met, and hence that the packetisation delay does not exceed a maximum threshold, the efficiency of outgoing cells may well be compromised. Hence this timer-CU plays a critical role in the multiplexing behaviour of AAL2.

- **Number of Users**

By providing multiplexing features at the AAL2 layer, multiple users can be supported. As the number of users increases, so does the efficiency. For a small number of users, a timer-CU is usually necessary to ensure PDUs do not wait too long. However, as the number of users is increased, a timer mechanism may not be necessary, as traffic enters the CPS transmitter at a fast-enough rate. Therefore one would want to establish how many users would be required to render a timer mechanism completely redundant, i.e. where 100 % efficiency could be achieved.

- **Packet Size**

The larger the packet size, the quicker the PDUs are filled. As a consequence the efficiency improves, and timer requirements become less critical.

- **Source Packet Distribution**

The faster packets are sent, the sooner PDUs are filled as well, hence allowing the same type of effects to be observed as in the above point.

To analyse the degree to which these metrics influence one another, the following assessments were performed:

- A PDU output analysis
- A PDU efficiency distribution test
- An analysis of multiplexing multiple users onto a single AAL2 path

6.8 PDU Output Analysis

If one ignores the delay-sensitive requirements for voice, then it is possible to make maximum use of the multiplexing capabilities of AAL2, as all outgoing PDUs are assumed to be full. For a given number of CPS packets, a series of AAL2 PDUs will be generated. The ratio of the number of AAL2 PDUs generated to CPS packets will vary according to the timer-CU value as well as the CPS packet size. When this ratio is high, say 0.8 or more, a large number of PDUs are generated for a given number of packets, and hence outgoing bandwidth is inefficiently utilised. However, the bandwidth utilisation can be optimised by selecting a suitable timer-CU value for a given CPS packet size.

When the effect of the timer-CU is introduced, PDUs are only partially filled before being sent out. The timer-CU, as well as the CPS packet size will have a direct effect on the PDU to packet ratio, and hence the reduction in outgoing AAL2 traffic that is achieved. To examine the relationship between these three parameters, three tests were performed. For each test, 200 packets were transmitted every 3 time units, over a total period of 600 time units. Furthermore each test consisted of five trials, where in each trial the value of the timer-CU for the AAL2 path was successively decreased by 3 times units. The results of these tests will now be discussed.

6.8.1 PDU Reduction with Small Packet Sizes

The CPS Packet size chosen for the first test was 10 bytes. A graph showing the five trials of this test, each plotted as an individual series, is shown in Figure 6.13.

The relationship between the various series of the graph is immediately clear, and quite expected. As the timer-CU was gradually decreased, the number of outgoing PDUs subsequently increased. In other words, the ratio of useful data to padded data decreased as the timer-CU decreased, hence increasing the amount of output traffic produced. The

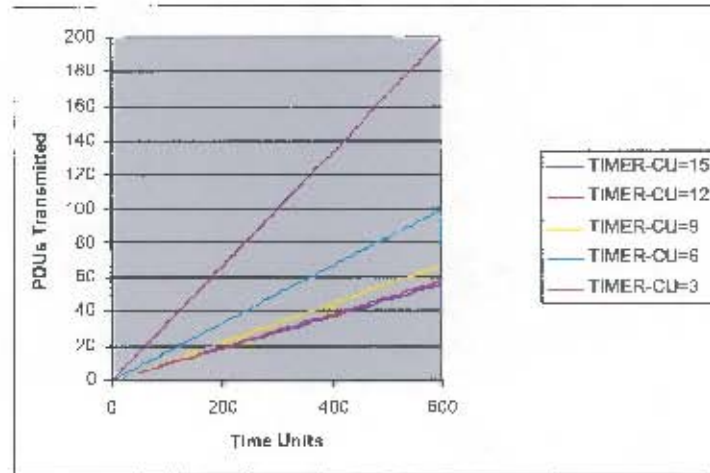


Figure 6.13: Number of PDUs transmitted for 10-byte Packets

intercepts on the right-hand vertical axis (at 600 time units) for all 5 trials are discussed below:

- A timer-CU of 15 corresponded to a total of 55 PDUs transmitted and hence a relatively high multiplexing gain that was achieved. In this instance the value of the timer was too large to have any effect at all. This was due to the fact that PDUs always departed before the timer had had a chance to interrupt the system. Being 10-byte CPS packets, by the time the 4th packet was appended to the partially filled PDU, at 12 time units, the PDU had already been dispatched. The number of PDUs transmitted may be calculated as follows:

$$\begin{aligned}
 \text{Number of PDUs transmitted} &= \frac{\text{Packet Size} \times \text{Number of Packets}}{\text{Max PDU Size}} \\
 &= \frac{(10+3) \times 200}{47} \text{ PDUs} \\
 &= 55 \text{ PDUs}
 \end{aligned}$$

where the total packet size including the header was 13 bytes.

- A timer-CU of 12 time units implied a marginally higher output PDU count. However, the manner in which the traffic was distributed across outgoing PDUs was quite different in that particular instance. Refer to Figure 6.14. Here, instead of each outgoing PDU being completely utilised, a phenomenon was experienced where the first

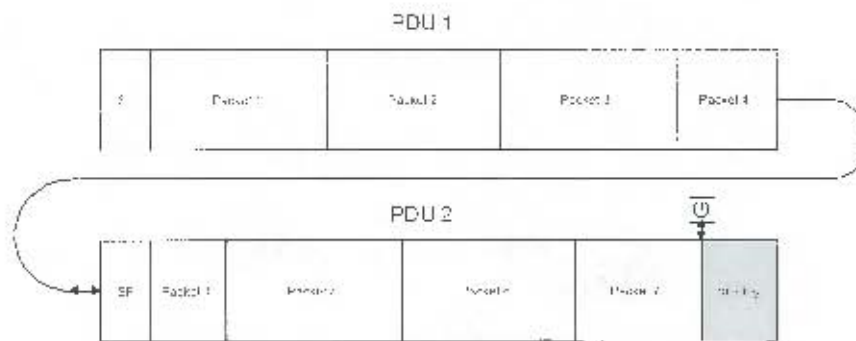


Figure 6.14: 2 PDUs Required for Every 7 Packets Transmitted

PDU was 100 % utilised and the second PDU was only partially filled before being dispatched.

As the inter-arrival time of arriving packets was set to 3 time units, the timer naturally only interrupted the system after four packets had been encapsulated. However, at the end of PDU 1, the timer was reset to 12 time units and the remainder of packet 4 was placed at the beginning of the second PDU. After three additional packets had been submitted, viz. packets 5, 6 and 7, the end of the PDU had not yet been reached, and hence the system was interrupted by the timer, as shown in the figure. This pattern was repeated in the 3rd and 4th PDU, and in the 5th and 6th PDU, and so on. The result was that for every 7 packets submitted, 2 PDUs were required. Therefore the number of PDUs required for 200 packets was simply calculated as follows:

$$\begin{aligned}
 \text{Number of PDUs transmitted} &= \frac{2 \times \text{Number of Packets}}{7} \\
 &= \frac{2 \times 200}{7} \text{ PDUs} \\
 &= 57 \text{ PDUs}
 \end{aligned}$$

- The following vertical intercept occurred at 67, where the timer-CU was set to 9 time units. After 3 CPS packets had been encapsulated in the PDU, the timer-CU expired. Hence every PDU transmitted only contained 39 bytes out of 47 possible bytes of useful information. Therefore the number of PDUs required was simply:

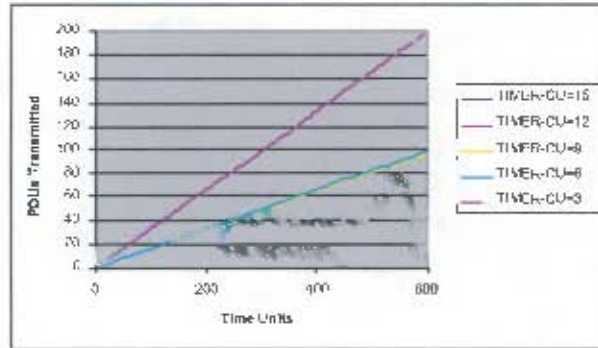


Figure 6.15: Number of PDUs transmitted for 20-byte Packets

$$\begin{aligned}
 \text{Number of PDUs transmitted} &= \frac{\text{Number of Packets}}{3} \\
 &= \frac{200}{3} \text{ PDUs} \\
 &= 67 \text{ PDUs}
 \end{aligned}$$

- When the timer-CU was set to 6 time units only 2 packets were multiplexed into each outgoing PDU. Hence half the number of PDUs to the number of CPS packets were required. Therefore, as 200 CPS packets were fed into the multiplexor, only 100 PDUs had to be generated.
- The lowest efficiency achieved occurred when only 1 packet was placed in the first 13 bytes of all PDUs, when the timer-CU was at its lowest possible value of 3. This resulted in the remaining PDU octets being filled with padding information. Therefore, for every packet transmitted, an equivalent PDU was required, resulting in a total of 200 PDUs that were transmitted.

6.8.2 PDU Reduction with Standard Packet Sizes

The parameters for the second test were identical to the first test; however here the length of the CPS packets were doubled to twenty bytes³. The results of the test are shown in Figure 6.15 and are explained below:

- When the timer-CU was set to either 15, 9 or 6 base units the same number of PDUs

³Note that the full packet size including header will now be 23 bytes - hence the full packet size is slightly less than doubled.

were produced. This is due to the fact that in all three cases, PDUs were completely filled before the timer could interrupt the system. Therefore lines representing these three trials lie directly on top of one another in the graph shown in Figure 6.15, and appear as a single series of points. The number of PDUs generated was calculated as follows:

$$\begin{aligned}
 \text{Number of PDUs transmitted} &= \frac{\text{Packet Size} \times \text{Number of Packets}}{\text{AAL2 PDU Size}} \\
 &= \frac{(20+3) \times 200}{47} \text{ PDUs} \\
 &= 97 \text{ PDUs}
 \end{aligned}$$

The number of additional PDUs required is easily calculated when these PDUs are 100 % utilised. This is shown below, where PIR represents the packet increase ratio:

$$\begin{aligned}
 \text{Additional PDUs required} &= \text{PIR} \times \text{Number of PDUs transmitted} \\
 &= \frac{(20+3)}{(10+3)} \times 97 \text{ PDUs} \\
 &= 97 \text{ PDUs}
 \end{aligned}$$

- A timer-CU of 6 base units for 20-byte packets implied that for every 2 CPS packets submitted, the timer would expire. In such an event the PDU contained 46 used bytes out of 47. Therefore, although one PDU was required for every 2 packets sent, each PDU was very efficiently utilised. Once again, the number of PDUs required was simply half the number of packets, i.e. 100.
- Only one packet could be placed in each PDU when the timer-CU was set to 3 base units. 200 PDUs, which was equivalent to number of packets inputted, was required.

6.8.3 PDU Reduction with Large CPS Packets

In the final test, the packet size was increased to 40 bytes. The effect of this operation is shown in Figure 6.16 below.

- In the figure, only two distinct trends were noticed. The first line intercepted the vertical axis at 183, and corresponded to the cases where the timer-CU was set to 15, 12, 9 and 6. In all four cases the timer was once again too slow to lower the PDU efficiency under 100 %. Once again, the number of PDUs required was simply derived from:

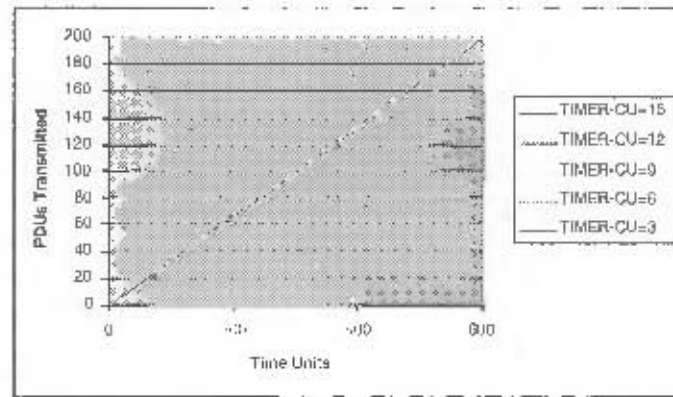


Figure 6.16: Number of PDUs transmitted for 40-byte Packets

$$\begin{aligned}
 \text{Number of PDUs transmitted} &= \frac{\text{Packet Size} \times \text{Number of Packets}}{\text{AAL2 PDU Size}} \\
 &= \frac{(40+3) \times 200}{47} \text{ PDUs} \\
 &= 183 \text{ PDUs}
 \end{aligned}$$

- In the second instance, one PDU was transmitted for every input packet received, resulting in a total of 200 PDUs that were transmitted.

6.8.4 Summary

The following may be concluded:

- The value of the timer-CU had a definite effect on the number of PDUs required for a given number of CPS packets, and the value of this timer was found to be inversely proportional to the number of PDUs transmitted.
- The larger the packet size, the more the timer-CU could be relaxed. As indicated by the three tests, increasing the packet size to 20 allowed one to relax the timer by 76 %, and increasing the packet size to 40 byte allowed a further reduction of 89 % to be achieved, yet in both cases still achieving 100 % efficiency.
- As the packet size increased, the difference between the minimum and maximum number of PDUs transmitted, based on the corresponding timer values, became gradually

less and less. In the first test, where the CPS packet size was only 10 bytes, the difference in the number of PDUs required was 145, whereas in the final test using 40-byte packets, the difference was only found to be 17. What may be concluded therefore, is that for small packet sizes, the effect of the timer-CU was significantly greater than that experienced for larger packets.

6.9 The PDU Efficiency Distribution

The PDU efficiency distribution gives a graphical indication of the payload utilisation of outgoing PDUs. Five tests were performed, where in each test the effects of packetising information generated from a particular codec with several timeout values were compared. The three codecs used in these tests were the following:

- G729 producing 10-byte packets, one every 10 time units,
- G729 producing 20-byte packets, one every 20 time units, and
- GSM full-rate, where 36-byte packets were produced every 20 ms.

The following large packet sizes were also taken into account:

- 44-byte packets. Together with the header, the packet size comes to 47, which fills the PDU payload exactly. Therefore one would expect to see a 100 % efficiency distribution for all such packets, as these packets are exactly PDU-aligned.
- 45-byte packets are just one byte too big to be handled by a single PDU, resulting in the last byte⁴ overlapping into the adjacent PDU. Depending on the affect of the timer, this could result in tremendous under-utilisation of PDUs, as will be seen in the subsequent tests.
- 64-byte packets are the largest possible packet size that could be accommodated by the CPS functionality of AAL2, due to the LI only being 6 bits wide, and hence efficiency distributions produced by such packets are of interest.

In each test below, 20 CPS packets were generated, and the efficiency for various timeouts for each test were analysed.

⁴This is provided, of course, that the 45-byte packet began at the first byte in the PDU payload.

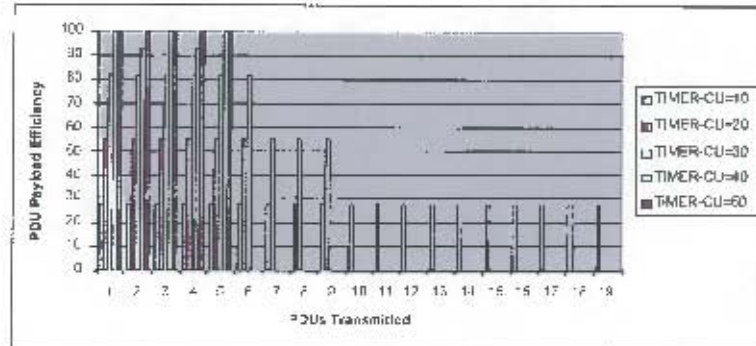


Figure 6.17: Efficiency Distribution for 20 G729-10 CPS Packets

6.9.1 Distribution with Small CPS Packets

The PDUs created from 20 CPS packets, each packet 10 bytes long, conforming to the G729 specification, are displayed against efficiency in Figure 6.17 below:

The following was noted:

- For a timer-CU of 10 (shown in blue), one PDU per packet was generated, resulting in the same number of PDUs being generated as packets. Hence, a very low PDU efficiency of 28 % was achieved, and was also calculated as follows:

$$\begin{aligned}
 \text{PDU Efficiency} &= \frac{\text{Occupied Bytes}}{\text{Maximum PDU Payload Size}} \\
 &= \frac{(10+3)}{47} \\
 &= 28 \%
 \end{aligned}$$

- When the timer was set to 20 time units (shown in maroon), one PDU for every two packets was generated, resulting in only 10 PDUs being generated and each PDU holding an efficiency of 55 %. The same effect was further noticed for a timer of 30 (shown in yellow), but once again only 6 PDUs were produced, each with an efficiency of 83 %.
- An interesting effect was observed with a timer of 40 (shown in cyan). Here it was noted that the efficiency for the first PDU was 100 %, whereas that for the second PDU was only 94 %, with this same pattern alternating in subsequent PDUs. The reason for such behaviour is straightforward and was explained earlier. The first

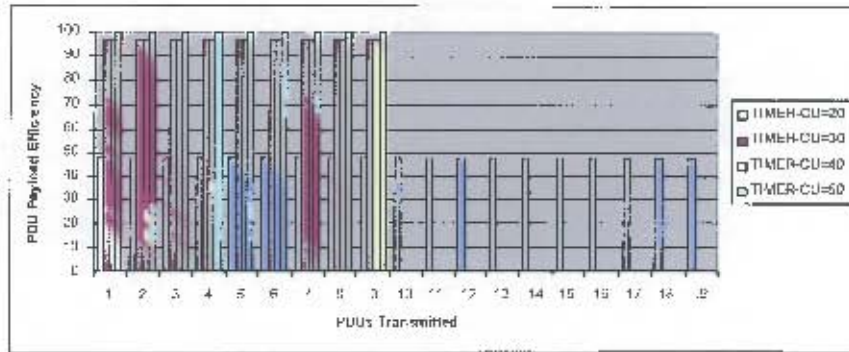


Figure 6.18: Efficiency Distribution for 20 G729-20 CPS Packets

PDU was found to be completely utilised and the remaining 5 bytes of the fourth packet were placed at the beginning of the following PDU. After three more packets were successfully placed in this PDU, the timer fired. The efficiency for this PDU was then simply:

$$\begin{aligned}
 \text{PDU Efficiency} &= \frac{\text{Occupied Bytes}}{\text{Maximum PDU Payload Size}} \\
 &= \frac{5 + (10+3) + (10+3) + (10+3)}{47} \\
 &= 94\%
 \end{aligned}$$

Furthermore, it should be noted that the overall efficiency is taken as the average efficiency over all PDUs transmitted, and is in this case equal to 97 %.

- Finally, with a timer of 50 (shown in magenta), all outgoing PDUs were fully utilised.

6.9.2 Distribution with Standard-size CPS Packets

The results of increasing the packet size to 20 bytes for the G-729 codec are shown in Figure 6.18 below.

The following was noted:

- PDUs were always found to be better utilised than the case for 10-byte packets, and the minimum efficiency increased to 50 %, as opposed to 28 %, regardless of any imposed timer effects.

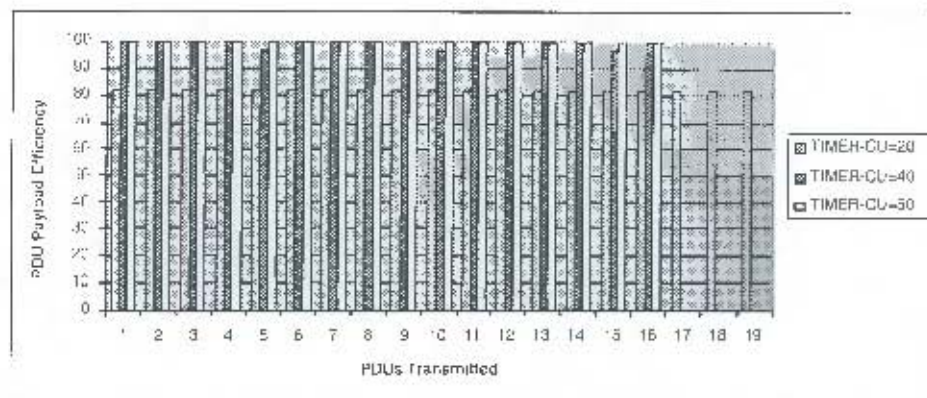


Figure 6.19: Efficiency Distribution for 20 GSM-FR CPS Packets

- No alternating efficiencies can be found, due to the way in which packets were distributed across PDUs.

6.9.3 Distribution with GSM-FR Packets

The efficiency achieved by plotting 36-byte packets conforming to the GSM full-rate specification is shown in Figure 6.19 below:

- Note the efficiencies achieved ranged from 82 % up to 100 %.

6.9.4 Distribution with 44-byte Packets

The maximum efficiency achievable is shown for the 44-byte packets in Figure 6.20 below:

- The distribution was found to be constant, as no matter what timer-value was specified, PDUs were always fully utilised. For this reason, only the effects of a single timer, implemented with the lowest possible value⁵, was shown.

6.9.5 Distribution with 45-byte Packets

The results for this test with two series of 45-byte packets with timers of 20 and 30 time units, are shown in Figure 6.21:

⁵The 44-byte, 45-byte and 64-byte packets are plotted with source distribution rates of 20 time units each.

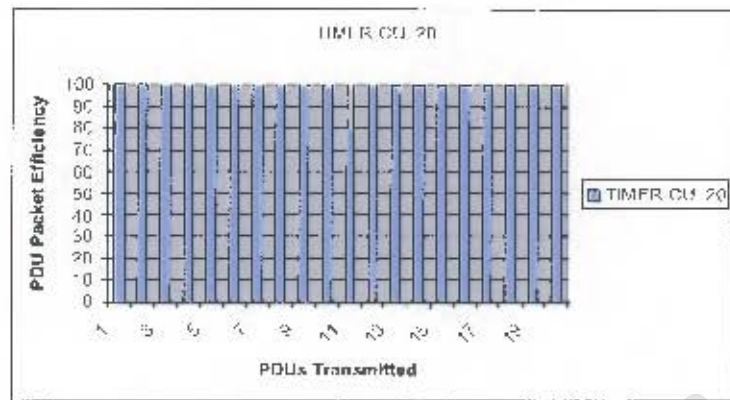


Figure 6.20: Efficiency Distribution for 20 44-byte CPS Packets

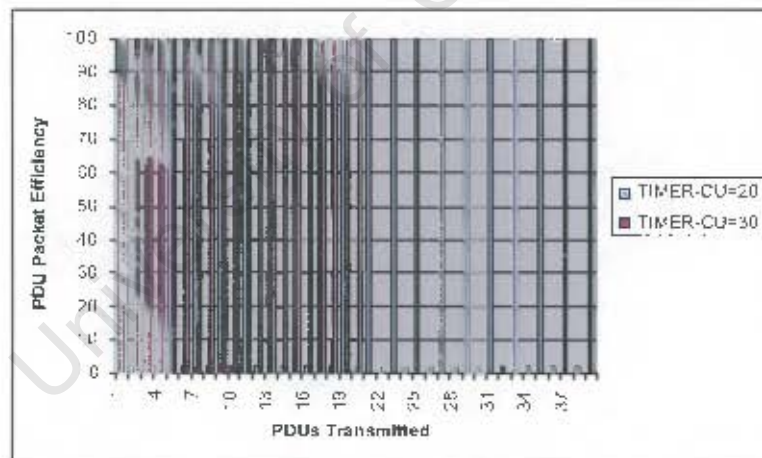


Figure 6.21: Efficiency Distribution for 20 45-byte CPS Packets

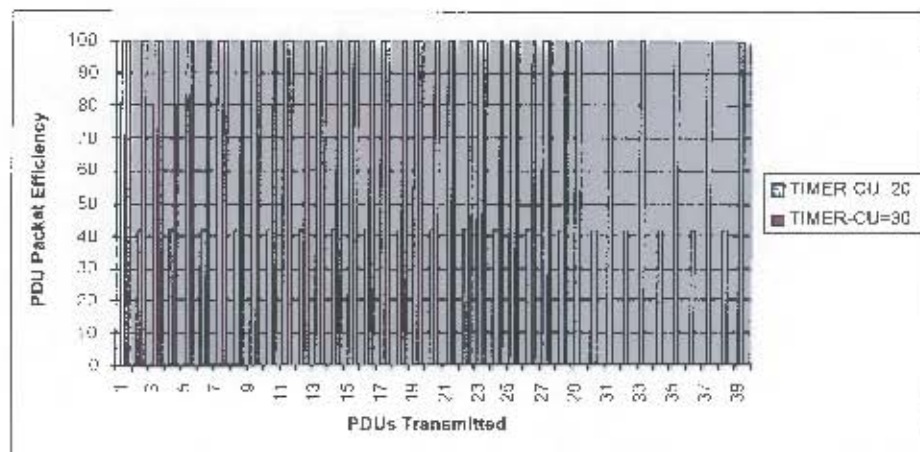


Figure 6.22: Efficiency Distribution for 20 64-byte CPS Packets

- Note the under-utilisation that occurred with a timer of 20 time units. Efficiencies were found to oscillate between 100 % and only 2 %, where only 1 byte out of 47 possible bytes was occupied by packet data. This was found to be the lowest possible efficiency that could occur in any given PDU. The overall efficiency over this range was therefore only 51 %.

6.9.6 Distribution with Large CPS Packets

The effect of 64-byte packets is shown in Figure 6.22 below.

- The same behaviour was achieved with 45-byte packets, apart from a better PDU efficiency.

6.9.7 Summary

In the graphs shown above, it is noted that only 19 PDUs were transmitted for each test, whereas 20 CPS packets were sent. This was because the CPS transmitter only dispatched a PDU when full or if the timer expired. By the time the last packet was transmitted, the logging function had already terminated.

The following diagram of Figure 6.23 shows the overall effect of the previous six tests:

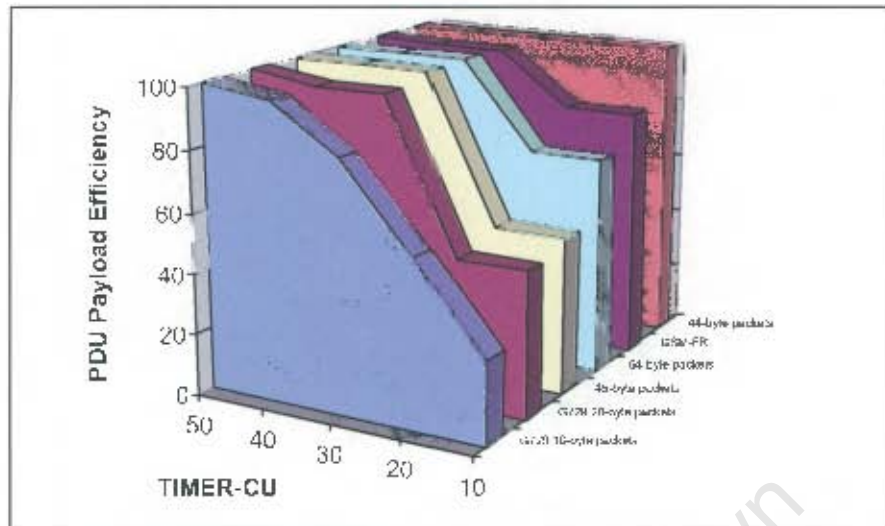


Figure 6.23: Efficiency Summary Graph

From this figure, the following was noted:

- Short packets with a rapidly expiring timer resulted in very low efficiencies, and would not generally be recommended.
- The average efficiency of a PDU distribution for the minimum possible timeout value was not necessarily proportional to the packet size. We saw above that although 64-byte packets were the largest, they ranked 3rd in terms of worst-case efficiency guarantees.
- It was always possible to achieve 100 % efficiency for any given packet size, by adjusting the timer accordingly. Obviously this would imply delay guarantees having to be sacrificed for a required efficiency being met.

6.10 Multiple Users

All the tests discussed thus far assumed packets arriving from a single source only. In the following series of tests, the number of users was gradually increased, and the efficiencies for the various packet sizes were compared. In each test, the effects on efficiency of up

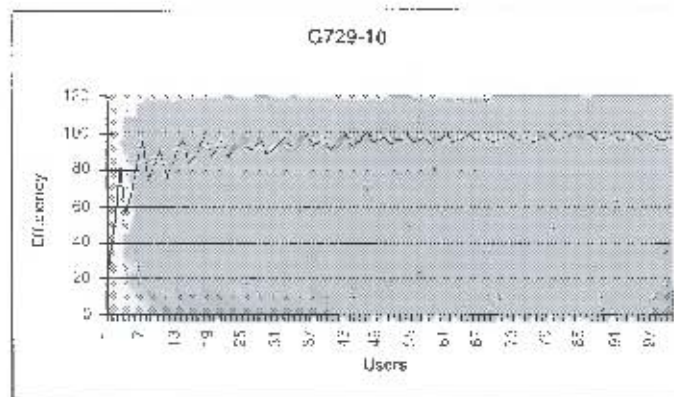


Figure 6.24: 100 users transmitting G729 10-byte Packets

to 100 users was analysed. Note that no test for 44-byte packets was conducted, as such packets had already been seen to always exhibit 100 % efficiency. The timer-CU for all tests below was set to the minimum value of 10 time units. Had the timer been larger, the outgoing PDUs would exhibit the same efficiency, regardless of the number of users. Therefore, to allow the number of users to dominate the results obtained, the timer-CU should be kept as small as possible.

6.10.1 Multiple Users Transmitting Small CPS Packets

100 users, each generating 10 byte CPS packets are analysed in Figure 6.24 above.

- Note how the efficiency began at 28 % for a single user and gradually increased.
- As the number of users increased, the oscillations between minimum and maximum efficiency gradually became smaller and smaller, until in the vicinity of 90 to 100 users, the efficiency varied between only 96 and 100 %. The reason for these dying oscillations was due to the PDU distribution that was produced for each run where the number of users was fixed. In the preceding section, we saw how the efficiency is not always the same for every PDU transmitted, but may in fact alternate between two values, with the same pattern re-occurring every two PDUs. When the number of users was increased, the same effect was observed, except that the probability of each outgoing PDU being 100 % utilised grew less and less. As efficiency was calculated as the average over a set of PDUs, it was found for example, that every

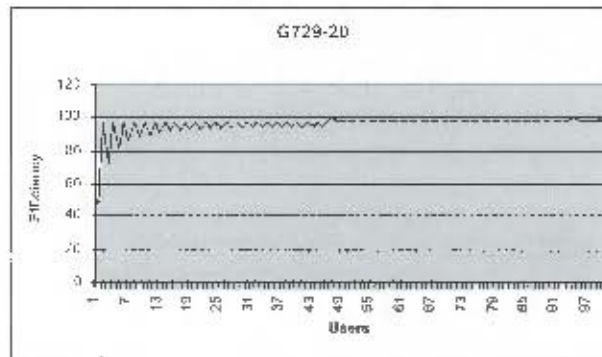


Figure 6.25: 100 users transmitting G729 20-byte Packets

four in five PDUs had an efficiency of 100 %, whereas the fifth PDU would only have an efficiency of 22 %. However, the average efficiency would be calculated as 84 %, as the efficiency of the last PDU would bring down the overall result. The larger this period becomes, the smaller the oscillations.

6.10.2 Multiple Users Transmitting Standard-size CPS Packets

The results for the second test are displayed in Figure 6.25:

- In the graph above an interesting phenomenon occurred. As the number of users increased, the minimum possible efficiency that could be achieved did not rise linearly, but at an exponentially decaying rate. What was believed to be actually decaying as the number of users increased, was the effect that a partial PDU will have on all other PDUs that were completely filled. When two users would transmit, the PDU would be almost filled up and the efficiency would be 97 %. However, when a third user was introduced, two PDUs would be required and the average efficiency would drop to 73 % (due to the second PDU only having an efficiency of 47 %). Therefore it can be seen that the low efficiency of the second PDU would have a dramatic effect on the first PDU and hence bring down the overall efficiency substantially, in fact, by 55 %. The next time this phenomenon would be observed would be when the number of users increased from 4 to 5, and when the efficiency dropped from 97 % to 81 %. However, in this case a drop in efficiency of 20 % would occur. In order to transmit data for 4 users, 2 PDUs would be required, and for 5 users, 2 and half PDUs would

be required. We saw that the third PDU had less effect in bringing down the total average. For a given set of PDUs, the effect that the single unfilled PDU will have on the total efficiency is what decays exponentially.

- Another point that is worth noting on the graph above is that the maximum achievable efficiency remained relatively constant at about 97 % until the number of users reached 47, where it jumped instantaneously to 100 %. At this point the maximum frequency jumped up to 99 % and remained constant until an additional increase at 94 users occurred. The reason for this behaviour was simply due to the fact that as 47 bytes were available for CPS packets in the AAL2 PDU payload (excluding the start field) packets were unlikely to exactly fill all outgoing PDUs until 47 sources existed. When the number of sources equalled the maximum transfer unit size, packet contents divided exactly into the series of PDUs, resulting in an efficiency of 100 %. This may also be shown in the formula below:

$$\begin{aligned}
 \text{PDUs Required} &= \frac{\text{Number of Sources} \times \text{Source Packet Size}}{\text{PDU MTU}} \\
 &= \frac{47 \times (20+3)}{47} \\
 &= 13 \text{ PDUs exactly}
 \end{aligned}$$

Note that 100 % efficiency was achieved when 47 users were transmitting simultaneously, regardless of the packet size, as the 47 in the numerator and denominator would cancel one another. This led one to conclude that the timer-CU was not required for 47 users or more, and had almost no effect. For the range from 1 to 10 users, the operation of the timer was critical.

6.10.3 Multiple Users with Large CPS Packets

The results for the remaining tests showing the efficiencies for multiple users for GSM-FR packets, 45-byte packets and 64-byte packets are shown in Figures 6.26, 6.27 and 6.28 respectively.

- For 45-byte packets there was scarcely any difference between the rates at which the minimum and maximum efficiencies rose, and the rate of both was seen to decay exponentially. The reason for this was that because the total CPS packet size was larger than the MTU of the PDU, efficiency levels could not drop below a threshold

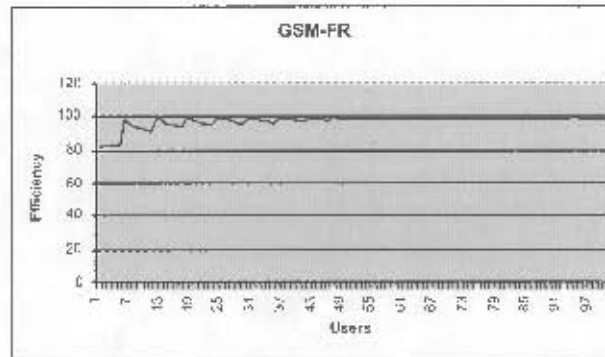


Figure 6.26: 100 users transmitting GSM-FR 36-byte Packets

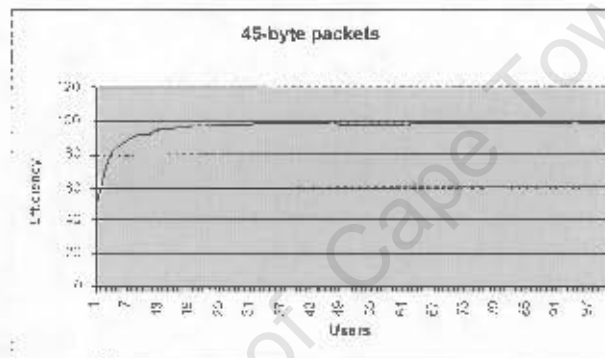


Figure 6.27: 100 users transmitting 45-byte Packets

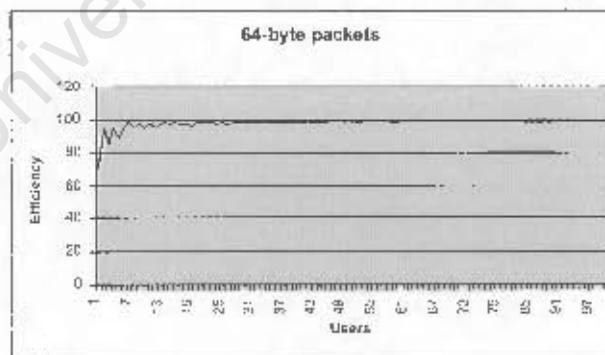


Figure 6.28: 100 users transmitting 64-byte Packets

once it had been reached, as it was impossible to fit a new packet in the remainder of the PDU.

- It may be said that GSM FR performs the best of all packets schemes (apart from 44-byte packets) as it was closest to the MTU⁶. The efficiency started with a relatively high value of 83 % for a single user and for 8 users was already found to oscillate in the 90-100 % band.
- The bumps experienced at higher user counts were simply due to errors introduced with rounding-off of values.

⁶Although 45-byte packets are strictly closer, they are larger than the MTU size and cannot be considered for comparison

Chapter 7

Conclusions

The preceding six chapters have presented the architectural design and implementation of an AAL2 switching node, where particular emphasis was placed on functions relating to the user-plane of the broadband ISDN model. It was shown how such a switch could be prototyped to match the sensitive and time-varying requirements of voice services, which can and are likely to be found in 3rd and 4th generation networks. By making use of the WUGS support architecture provided by Washington University, the development time required to achieve a working system was substantially reduced. Furthermore, the operating nature of the embedded system (SPC) and support end-stations, allowed for extensive testing to ensure switch compliance, as well as obtaining suitable performance benchmarks.

The implementation status of the AAL2 switch is at a stage where the following functions are operational:

- The AAL2 CPS, which is capable of converting CPS Packets to AAL2 PDUs and vice versa.
- AAL2 route establishment in the switch, and most importantly, switching based on the channel identifier (CID) of individual AAL2 sub-channels
- Efficient CID lookup engine, where modifications to AAL2 routes in the switch are only required when new channels are added, deleted or re-shuffled
- Support for multiple AAL2 input and output paths

- The ability of the PSU to distinguish between three different service classes, i.e. low delay and normal delay for voice, and high-delay for SSCS-data
- Support for common channel signalling (CCS) on CID 8, and ability to pass AAL2 signalling packets to the SPU (Signalling Processing Unit)

Due to the vast scope of issues imposed by AAL2 and AAL2 switching in the WUGS environment, careful consideration had to be given in the implementation phase as to what modules ought to be prototyped first, and of particular importance, how the development time of these modules could be reduced. The principle components of the design were implemented in a phased approach, and are presented in the order in which they were completed:

- The ADU (AAL2 Disassembly Unit) and ARU (AAL2 Re-assembly Unit) were the first modules to be completed, for the simple reason that these modules are not related in any way to AAL2 switching. It was therefore possible to fully test the operation of the AAL2 CPS by creating the AAL2 generator and analyser based on the same technology, connecting these two devices together¹, and monitoring their operation. By basing the AAL2 receiver and AAL2 transmitter mechanisms on the recommendation given by ITU-T 363.2, which form the core functions of the ADU and ARU respectively, the development time of these modules was kept to a minimum, allowing more focus to be placed on the AAL2 switching itself. Furthermore, the modular nature of these units and the use of threads required one to only have to consider the case for a single AAL2 path - the technology scaled very well to multiple input and output paths.
- The PSU (Packet Scheduling Unit), with its corresponding input and output packet stores, were the following components to be completed. By realising that the buffering requirements for both packet-stores could be fulfilled by multiple FIFO mechanisms, and implementing each delay class in the scheduler as a separate linklist, special attention was given to the development of separate FIFO and linklist algorithms. The primary purpose of these algorithms was to provide an abstract interface to these respective data types, that could be re-used where needed in the switch², hence

¹In reverse order, of course, with an ARU-like device (AAL2 generator) at the source to generate the packet stream and an ADU-like device (AAL2 analyser) at the receiver to receive and process the AAL2 traffic.

²Information on the various ADTs is provided in Appendix D.

providing a more comprehensive code-base and further reducing the development time.

- The following stages involved incorporating the CID translation engine into the ARU mechanism via the use of channel sets, re-using the features provided by the linklist ADT, as well as including channel tables in the ARU to access the central packet stores, and in this manner, making use of the FIFO ADT.
- The SPU (Signalling Processing Unit) was the following module to be implemented. Note that the intention of this research was not to produce a fully operational SPU, but instead provide a workable environment, with a well-structured interface, on which a higher-level AAL2 signalling framework, which could make use of extended AAL2 SSCS capabilities, could be implemented.
- The supporting framework for the CCU (Configuration and Control Unit) was prepared during the next phase. As the previous modules had now already existed, and the CCU was declared to be the first module to receive execution rights, the development of the project was at a stage where the CCU could instantiate these existing modules to commence the first testing procedures.
- Finally, the implementation of the timer unit and corresponding inverse priority heap ADT was completed, and their functionality was incorporated into the ARU. Due to the limitation imposed, extensive work was required at this phase to provide a working and elegant mechanism to provide multiple timers for multiple paths.
- Although incorporated into the design, the IU (Interrogator Unit) and ACM (Admission Control Unit) were not implemented.

The ability to extend a port of an ATM switch (WUGS) with an embedded computer (SPC) running the NetBSD operating system provided the developer with the resources to focus on open-source, higher-level algorithms, not achievable with many commercial platforms and proprietary software. However, this chosen environment did present several of its own drawbacks in the implementation phase:

- The internal processor of the SPC is clocked at a frequency of 166 Mhz, which is comparatively slow by today's standards. A logical side effect of this was the

limitation on the number of users that could be supported³. The system was further slowed down by general operating system house-keeping, which included memory management and device driver overheads, as well as the additional CPU consumption of the external GNU-Pthreads library.

- As NetBSD is not a real-time operating system, it was not possible to guarantee the scheduling and processing of traffic in real-time.
- A major side-effect that the SPC architecture had on the AAL2 switch, was first the fact that a maximum of only 1 timer could exist concurrently, with the additional imposed maximum resolution of this timer. The maximum resolution was found to be 10 ms. Even in cases where ample processing capability was available for a module to complete its task under 10 ms, there was no method known to the author at the time as to how a smaller time period could be measured. This required all results to be calibrated by normalising the time-scale to time units, where each time unit represented 10 ms.
- In Chapter 5, it was discussed how events could be passed to the SPU by means of message ports. This was required for example, when the ADU had detected a signalling packet, and had to notify the SPU accordingly. When multiple signalling packets occurred in a single PDU, multiple events would be required to be generated to the SPU. However, it was noticed that the GNU portable threads library did not support sending multiple messages to the same message port. Hence, for each message sent to a message port, the message had to be extracted by the SPU before a new message (for another signalling packet) could be sent. This drawback was solved by sending a pointer to a queue of signalling messages, as well as the number of messages to be extracted, for each CPS receiver run.

To evaluate the functional correctness of the AAL2 switch, several tests were conducted, first on two end-systems transmitting AAL2 packets to one another, and at a later stage on the switch itself. In all tests performed, a stream of packets or PDUs was sent, and the resulting switch or end-stations' actions were analysed and logged. By examining these logfiles off-line, it was concluded that both end-stations and switch were functioning correctly. Additional tests were performed to analyse the ability of the switch to expand

³The multiple user results of chapter 6 had to be implemented on a high-performance processor with APIC ports

traffic from a single input AAL2 path to multiple output paths, and to condense traffic from multiple input paths to a single outgoing AAL2 path. These tests were confirmed to be successful, as shown in Chapter 6. A final test to ensure operating conformance to the design was to ensure that all signalling packets interspersed with user traffic on a given input path would be retrievable by the SPU, which was found to be true.

By analysing in detail the AAL2 layer functionality itself, useful performance-related results were obtained. It was found that the value of the timer-CU on a given path affected the efficiency of outgoing traffic, particularly when short packets were transmitted. The timer-CU was found to be inversely proportional to the number of outgoing PDUs for a given number of input packets. This implied that relaxing the timer-CU somewhat, although perhaps sacrificing the quality of voice channels on the path, would result in a substantial reduction in the amount of output traffic produced. It was found that the codec used to generate voice had an effect on the performance of AAL2, and that the timer-CU, efficiency and codec itself were intimately related to one another. It was shown in Chapter 6 how 100 % efficiency could always be achieved, regardless of the packet size, by relaxing the timer-CU sufficiently, and that the minimum efficiency was not necessarily proportional to packet size, as what might have been previously envisioned. Finally, the effect of increasing the number of users was analysed. Of interest was to determine the point at which 100 % efficiency would be reached without the need for a timer. This test concluded that such an efficiency could be reached with a minimum of 47 users. It was also found that the point at which 100 % efficiency could be reached recurred when increasing the number of users, and was influenced to a large extent by the packet size.

Chapter 8

Recommendations

Although a functional AAL2 switch has been implemented, it was simply not possible to cover all foreseeable concerns and aspects of this technology, and there are hence many open-ended issues that yet still remain to be further researched and analysed. Many of the recommendations were only realised in later stages of the project, and several of these may in fact be transposed to possible future research projects.

Based on the preceding work, and conclusions drawn, the following recommendations can be made:

- The only sub-layer of AAL2 that was implemented is CPS, and hence SSCS functionality still remains to be investigated. Additional work would therefore be required to implement such an SSCS, where the additional functionality gained would be essential. As an example, AAL2 signalling specifications make mention of signalling packets that may exceed the maximum CPS allowable size of 64 octets, thus requiring this intermediate functionality. Furthermore, higher-level applications wishing to make use of AAL2 as a transport layer may find the CPS imposed packet limit a severe overhead drawback.
- The signalling framework to complete the control-plane functions of the switch must still be implemented and is currently underway. As the SPU will provide support for AAL2 signalling only, additional work would be required to enhance the switch with ATM signalling capabilities. This would require implementation of the Admission Control Module with a UNI 4.0 compliant signalling stack. Work in this field is also

currently underway to augment the WUGS with ATM signalling capabilities¹.

- Due to constraints of the SPC that were mentioned in the previous chapter, the logical next step to be taken would be to migrate several of the switch modules to a hardware platform, such as the Field Programmable Port Extender (FPX). The FPX is a specialised board provided by Washington University that will allow the functionality of a switch port to be extended by means of a synthesised logic design. Features that would be migrated to hardware would be those based at a low level, that are conceptually simple, and that require intensive processing capabilities, such as cell interrogation, packet extraction, fast CID lookups and timing. For these reasons, likely candidate modules for hardware migration would include the IU (although not implemented yet), certain functions of the ADU and ARU as well as the base timer unit². Modules that ought to remain in software would include those comprising fairly complex algorithms and those that perform system updates on a far more rare scale to their hardware-suggested counterparts. This include the SPU, ACM and CCU.
- Although implementations employing the usermode code of the APIC are likely to be more complex than socket based function calls, with a more in-depth understanding required of the APIC and its internal organisation and registers, it would be of interest at some stage to investigate the feasibility of such an implementation. It is strongly expected that the usermode code would operate more efficiently than a native socket interface. A wrapper interface could be written to hide the internal APIC details for future development, and in so doing, providing a socket-alive interface.
- Further research would still be required to investigate what features could be supported in the CCU to provide management features, in the form of a suitable management framework. The system operations and management function (OSF) would liaise closely with the various modules of the system and perhaps the user-flows themselves to collect fault and performance data for the switch. Throughput information could be collected in a similar fashion to be used for billing purposes. The OSF could furthermore accept commands from an out-of-band channel for controlling the

¹The only signalling protocol currently supported in the WUGS is CMAP, which a proprietary signalling protocol.

²The base timing could be achieved at a hardware level with extremely fine granularity, with a higher-level software wrapper to provide timer updates.

switch and generate information in the reverse direction, possibly along the same out-of-band connection, to a monitoring station for further analysis.

- Although ATM multi-casting is well understood and well-researched, little work has been done in the area of AAL2 multi-casting. With AAL2 multi-casting extensions in the switch, a channel on an input path can be range-copied to form multiple outgoing channels. The author wishes to define intra-path AAL2 multi-casting as the copying of channels that will all exit the switch on the same AAL2 output path. Inter-path AAL2 multi-casting may then be defined as the copying of channels that will exit the switch on two or more outgoing paths.
- As NetBSD is not a real-time operating system, it would be of interest to port the switch architecture to a platform with real-time capabilities. There are literally dozens of real-time operating system available on the market today, and a fair amount of research would be required to select a suitable OS. However, two possible real-time platforms that may be worth investigating, and which are both open-source, are Q-Linux and RedHat's eCos for embedded systems.
- It is well known that switching AAL2 packets at an early stage in the network would in most cases reduce congestion. Therefore, placing AAL2 switches at various locations in an ATM network, where the positions of these switches haven't been thought out carefully, may well result in a sub-optimal AAL2 switching topology. Therefore, given an interconnected mesh of ATM switches, where a number of AAL2 switches are to be added, it should be possible to recommend positions for such switches, as well as the relative density for these switches in various regions of the network. The optimum solution will of course depend on factors such as average network load on various links, latency, number of ATM switches as well as nature of the traffic being transported.
- When a WUGS is fully populated with AAL2 modules, the load at a given instance in time will not be constant across all ports. What is proposed therefore is an intelligent AAL2 master that can interact with the various AAL2 slaves, which themselves are responsible for the switching. The AAL2 master will be responsible for examining the load on each port to determine which modules are overloaded and which modules are under-utilised. By redirecting AAL2 traffic from overloaded modules to under-utilised modules by means of recycling, it should be possible to achieve a certain

level of load balancing. A neural network may further be used to “predict” oncoming congestion, where contingency measures could be made.

University of Cape Town

Bibliography

- [1] K. Ahmad, *Sourcebook of IP and ATM Interworking*. Wiley Interscience, 2002.
- [2] C. R. Retreat, "Who is the inventor of atm?," <http://cell-relay.indiana.edu/cell-relay/FAQ/ATM-FAQ/g/g4.htm>, 1997.
- [3] E. S. Oscar Mezquita Baeza, "Performance evaluation of atm/aal2 as switching technology in 3g mobile access network," 2001.
- [4] D. R. Brian Kernighan, *The C Programming Language*. Prentice Hall, 1978.
- [5] T. Chaney, "Design of a gigabit atm switch," *IEEE Communications*, May 1997.
- [6] S.-C. H. P.-C. W. Yaw-Chung Chen, Chia-Tai Chan, "Traffic control approaches for voice over atm networks," *IEICE Trans. Communications*, vol. E81-B, pp. 2380–2391, December 1998.
- [7] Z. D. Ditta, *APIC User's Manual*. Applied Research Laboratory, 1998.
- [8] C. S. K. S. T Enderes, S.C. Khoo, "Impact of statistical multiplexing on voice quality in cellular networks," *ACM MSWiM 2000*, pp. 028–035, August 2000.
- [9] G. L. A. R. I. S. Goran Eneroth, Gabor Fodor, "Applying atm/aal2 as a switching technology in third-generation mobile access networks," *IEEE Communications Magazine*, pp. 112–122, June 1999.
- [10] W. University, "Gigabit kits website," 1999.
- [11] R. M. A. O. U. D. Sandra Loosemore, Richard Stallman. Free Software Foundation, 1999.

- [12] L. Greenstein, "Transporting voice traffic over packet networks," *International Journal of Network Management*, vol. 8, pp. 227–234, 1998.
- [13] L. M. M. S. N. Antonio Manuel Grilo, Pedro Miguel Carvalbo, "Vtoa/voip/isdn telephone gateway," *2nd Intl. Conf. on ATM (ICATM'99)*, pp. 230–235, January 1999.
- [14] T. O. k. M. Masahide Hatanaka, Toshihiro Masaki, "Vlsi architecture of switching control for aal2 type2 switch," *IEICE Trans. Fundamentals*, vol. E83-A, pp. 435–441, March 2000.
- [15] ITU-T, "B-isdn atm adaptation layer specification: Type 2 aal," September 1997.
- [16] K. M. Mike McLoughlin, "Adapting voice for atm networks - a comparison of aal1 vs aal2," *White Paper*, 1997.
- [17] V. Moondhra, "Implementation and performance analysis of atm adaptation layer type 2," Master's thesis, B.E. University of Mysore, India.
- [18] N. Networks, "Next-generation packet telephony networks," *White Paper*, February 2001.
- [19] P. S. Y.-Q. L. David W Petr, Raghushankar R Vatte, "Efficiency of aal2 for voice transport: Simulation comparison with aal1 and aal5," *Intl. Conf. on Communications (ICC'99)*, June 1999.
- [20] E. J. Rook, "A case study of an approach to the deployment of atm and ip into an international telephony network," *2nd Intl. Conf. on ATM (ICATM'99)*, pp. 276–284, June 1999.
- [21] H. Saito, "Performance evaluation of aal2 switching nodes and a network using them," *Intl. Conf. on Communications (ICC'99)*, 1999.
- [22] K. Sriram and Y. Wang, "Voice over atm using aal2 and bit dropping: Performance and call admission control," *IEEE*, pp. 215–224, May 1988.
- [23] S. D. Baranitharan Subbiah, "Atm adaptation layer 2(aal2) for low bit rate speech and data: Issues and challenges," *IEEE ATM Workshop*, pp. 225–233, May 1998.
- [24] K. M. Hiroshi Sunaga, Takenori Oktunai, "Large-scale vtoa switching node architecture," *IEICE Trans. Communications*, vol. E82-B, pp. 70–79, January 1999.

- [25] H. T. Fumio Suzuki, "Third generation mobile telecommunication system," *IEICE Transactions*, vol. E82-C, pp. 1871–1878, November 1999.
- [26] A. S. Tannenbaum, *Modern Operating Systems*. Prentice Hall, 2nd edition ed., 2001.
- [27] J. S. Turner, "Queuing analysis of buffered switching networks," *IEEE Communications*, vol. 41, no. 2, 1993.
- [28] J. S. Turner, "Architectural choices in large scale atm switches," *IEICE Transactions in Communications*, vol. E81-B, pp. 120–137, February 1998.
- [29] D. Wright, "Voice over atm - an evolution of network architecture alternatives," *IEEE Network Magazine*, September/October 1996.
- [30] R. O. S Wright, "Ip "telephony" vs. atm: What is there to discuss?," *IEEE Intl. Conf. on ATM (ICATM'98)*, pp. 400–409, February 1998.

Appendix A

Traditional Methods for Transporting Voice in ATM and SSCS Extensions

A.1 Introduction

Although AAL2 is superior at carrying short voice packets for variable bit-rate delay-sensitive applications, AAL1 as well as AAL5 may still be used to carry voice under special circumstances, or when AAL2 is simply not available. This appendix will briefly examine these two alternative adaptation layers when used to transport voice.

This appendix will also provide a brief outline of additional functionality that may be achieved by means of a Service Specific Convergence Sub-layer (SSCS) implemented on top of the standard CPS.

A.2 Voice over AAL1

The use of AAL1 for carrying voice over ATM was standardised by the ITU-T in 1993. The mechanism by which PCM-based voice may be carried over an ATM connection was defined by the ATM Forum as Circuit Emulation Service (CES). There are two methods for placing voice channels over an AAL1 connection:

- Unstructured circuit emulation, refers to the process of placing the information from an entire E1 or T1 pipe onto an ATM connection, without regard to the sub-channels

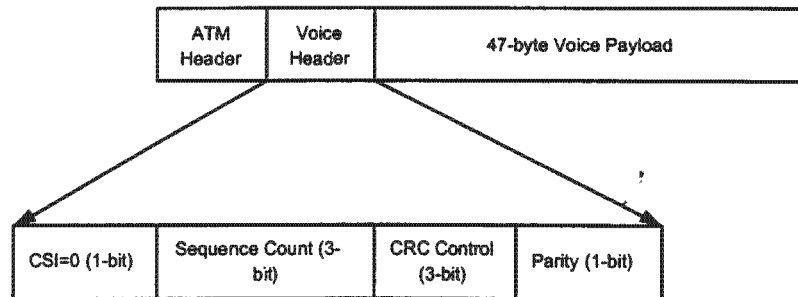


Figure A.1: Encapsulation of Voice onto AAL1 Payload (Adapted from ATM Forum)

within the data pipe. The disadvantage of this scheme is that the total E1/T1 bandwidth is permanently allocated, even when some of the sub-channels are idle.

- Structured circuit emulation, refers to the process of carrying individual sub-channels that belong to an E1 or T1 connection. Hence only the accumulative bandwidth that is required for these N channels need be reserved.

AAL1 does not support variable bit-rate voice, and is therefore best suited to the transport of uncompressed, synchronous voice, such as PCM encoded samples from a G-711 vocoder. An AAL1 connection would therefore be valuable as a CBR trunk between two digital PBXs.

AAL1 is suitable for the transmission of voice as it provides synchronisation between the source, network and receiver. Synchronisation is an essential requirement to ensure voice samples can be played out at the same rate at the receiver as was sent by the source. The method for encapsulating voice into the AAL1 payload is straightforward, and is shown in Figure A.1.

Voice samples are placed into the 47-byte payload of the AAL1 frame. A voice header is appended to provide sequencing if required. Finally, the ATM cell header is appended and the cell is dispatched.

AAL1 cannot provide the same multiplexing gain as what may be achieved by AAL2. However, there are certain instances where AAL1 should be used instead:

- When inter-working with N-ISDN networks, AAL1 would be required for synchronisation with the network clock.

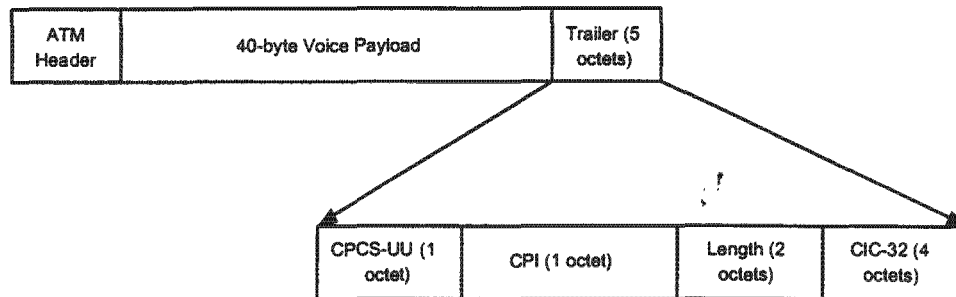


Figure A.2: Encapsulation of Voice onto AAL5 Payload (Adapted from ATM Forum)

- AAL2 provides bandwidth efficiency only for compressed traffic streaming at a low bit rate, and where excessive packetisation delay would mar the quality of the voice. However, the packetisation delay achieved for G-711 based voice onto an AAL1 PDU is only 5.8 ms, and hence AAL1 is quite adequate for such voice. Furthermore making use of AAL2 would prove to be inefficient.
- Voice services, such as fax and data, do not operate well over an AAL2 connection, and hence one should consider making use of AAL1 in this case.

A.3 Voice over AAL5

Although AAL5 was initially intended for data application, it may indeed be used for voice, and operates in a similar manner to AAL1. The difference with AAL5 is that only 40 octets may be used for voice data as shown in Figure A.2, instead of 47 as in AAL1. Furthermore, an 8-byte trailer is appended to the voice information to construct a full PDU.

A.4 SSCS Extension to AAL2

Throughout this text, we have assumed the absence of an SSCS layer. Two SSCS specifications have been defined by the ITU-T thus far:

- I366-1 was defined to allow larger than normal packets to be carried over an AAL2 connection. This functionality is required for message passing via the AAL2 net-

work and was not intended for real-time applications such as voice. AAL2 signalling messages and certain data applications would benefit from this specification.

- I366-2 defines the mechanisms whereby circuit-mode data, voice-call information and in-band signalling messages may be carried over an AAL2 connection. Three types of packets are defined:

- With Type 1 packets, no error checking is performed on the payload whatsoever.
- Type 2 packets are partially protected with a 5-bit cyclic redundancy check (CRC)
- Type 3 packets are fully protected and are appended with a 10-bit CRC field. Type 3 packets are used to carry control data, such as fax control information (not the image), E-164 addressing information and signalling data.

Appendix B

SDL Nomenclature

B.1 Introduction

The symbols used for the design specification of chapter three are explained here. Note that the symbols used in the design specification form an extremely small subset of available SDL symbols.

The various SDL symbols that were used are shown in Table A2.1 on the following page.


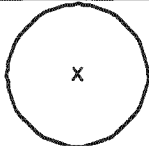
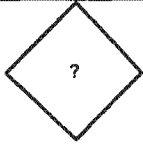





SDL Symbol	Meaning
	Action
	Connector
	Decision
	Input Signal / User Request
	Output Signal / User Indication
	Procedure
	Start
	State

Table B.1: SDL Symbols used and their Meaning

Appendix C

AAL2 Switch Evaluation Test Suite

C.1 Introduction

This chapter will describe the test suite that was developed and used to evaluate the performance and functionality of the base AAL2 switch. The test suite was comprised of two components, as introduced in chapter 6. The AAL2 Generator was responsible for emulating the source traffic of multiple AAL2 sources, where each source's attributes could be individually specified. The AAL2 Analyser was responsible for receiving AAL2 traffic from the network, (which may or may have originated from an AAL2 switch), extracting the individual packets, and logging these packets to file for further analysis. Note that the name "Analyser" for this AAL2 receiver may be slightly misleading and cause one to think that analysis is only performed at this particular end-point in the network. However, analysis, particularly when obtaining performance benchmarks, was also performed at the AAL2 Generator itself, where the timer-CU was implemented, and the reader's attention is drawn to this fact. This appendix will briefly discuss the internal operation and organisation of these evaluation tools.

C.2 The AAL2 Generator

The AAL2 generator was based on a non-threaded architecture, and implemented as a monolithic unit. There were two reasons why the support given by threads was not required for this generator:

- It was not the intention to include the emulation of multiple AAL2 paths into the generator, but simply the function of being able to multiplex multiple channels onto a *single* AAL2 path. It will be seen shortly how a single generator can be re-spawned as multiple processes to allow for the streaming of multiple input AAL2 paths into a single AAL2 switch.
- In comparison to the switch, the AAL2 generator may be likened to the ARU, as it is primarily responsible for the creation of AAL2 traffic. No support for signalling, management, or scheduling was required. Due to the processing power required by threads, it was decided at the testing stage to rather dispense with this thread-based environment and allocate more processing power to the multiplexing of multiple sources.

The generator software was written in the same environment as the switch itself, and a large portion of the ARU code was reused, apart from the timer mechanism, whose operation will be explained shortly. Before the operation of the generator is described, it would be helpful to understand the manner in which sources are defined. Data describing the various sources is defined in a file known as *sources.conf*, which must be placed in the same directory as the generator client program. If the file is re-edited, the generator must be stopped and restarted before the modified file can be read in.

```
#source configuration file - no whitespace!
#global options
timer-cu = 50
aal2-path = 90
sequence = 1
#source specific options
#---Source 1---
define source
begin
name = theCompressedSource
codec = g729
packet-length = 10
packets = 12
aal2-channel = 18
active = 1
it = 10
logfile = s1.sd
end
```

As is noted, there are two main sections of the file, viz. global options and source specific options. Global options relate to the entire AAL2 path onto which channels are multiplexed and include the following definitions¹:

- Timer specification (*timer-cu*), defines the timer-CU in base time units for the path. Note that the timer-CU should not be less than the lowest source inter-arrival time (*it* - discussed below).
- AAL2-Path Identifier (*aal2-path*), defines the VC that ought to be allocated for the outgoing path, and must match an already open channel bound to the APIC NIC².
- Sequence Number status (*sequence*) specifies whether the sequence number, as discussed in Chapter 2 on AAL2, should begin with a 1 or a 0.

Source specific options are used to define the attributes for an individual source. A “source block” is begun by the tokens *define source* and *begin*, followed by various options, and must be terminated by the *end* token. A maximum of 20 source blocks may be currently defined, but this limit is easily extended. As shown in the file example above, a source should be described in the following manner:

- Source name (*name*) is a canonical name given to a source and may be used simply to aid the tester.
- Source codec (*codec*) defines the source according to one of the following codecs: g-726, g-727, g-728, g-729, gsm-fr and gsm-hr. By specifying one of these codecs the packet rate and packet size will be automatically specified. For example, specifying the source codec as gsm-fr will define a packet length of 36 bytes and an inter-arrival packet time of 20 time units (equivalent to the 20 ms specified for this codec). However, the user of the generator is discouraged from using this option, and should rather explicitly define the codec parameters.
- Length of Packet: (*packet-length*) is used to define the length of individual packets explicitly, and is the preferred method for specifying packet sizes. Note that such a length, in the context of AAL2, will exclude the CPS packet header.

¹Tokens in italics indicate the syntax to be used in the configuration file.

²The process of binding a channel to the APIC NIC is discussed in Appendix E.

- Number of Packets (*packets*) specifies the number of packets to be generated. Once the allocated quota for a source is reached, the active bit (defined below) is cleared and the source will automatically become inactive.
- Source Identifier (*aal2-channel*) defines the allocated CID to be stamped into the header of CPS packets as they are generated. Note that fake CCS signalling packets may be created simply by specifying a CID of 8.
- Active Bit (*active*) declares whether a source should start up in the active mode or not, and should generally be set to 1. Its intended use is to allow a source to be deactivated without having to remove or comment out the source description from the source file.
- Packet Inter-Arrival Time (*it*) specifies the inter-arrival time of packets of the particular source in base time units³.
- Logfile (*logfile*) sets aside a log file to which packets will be written when they are generated. Note that when multiple sources are defined, the user should remember to allocate a separate log file, with a unique name, for each source.

Note that the syntax of the source file is extremely strict and the parser is sensitive to whitespace in the regions where it shouldn't be. If tokens are not defined as shown in the source file, they will be ignored.

The operation of the AAL2 generator may now be described as follows:

- The first operation to be undertaken by the generator is to parse the source configuration file, *sources.conf*, and store the source data in a source attribute array, where each attribute entry defines a single source.
- In the following stage, the actual packet data is generated. For each source, the specified number of packets, each the correct size, are generated with random payloads. A CPS header with the specified CID value is appended to the packets and logged to the specified file. Once this stage is complete, there should be N individual packet files, if N sources were defined in the configuration file.

³Currently, only sources of the CBR-traffic class can be generated.

- Now that all packets to be generated are on file, and are hence safely stored for later analysis, the generator proceeds to read all packets into memory. As there is ample memory to store sufficient packets for the various tests that would be required, it was decided to rather stream packets from memory than from secondary storage, simply due to the excessive delays that would be experienced in reading each packet for a file.
- The final pre-processing stage involves initialising an output AAL2 path onto which the AAL2 traffic may be placed, initialising the transmitter state machine to default values, resetting the timer reference frame to zero and opening any logfiles to store statistics during the generator run.
- The main loop of the AAL2 generator is driven by the timer handler function. When this loop is first entered, it will wait for a global variable to be cleared, which signifies that a timeout has occurred. The timeout can indicate that a source packet must be sent, or that the path timer has expired. If the required inter-arrival time for a source has been reached, and the source is an active source, then an additional packet from the logfile matching the source will be submitted to the CPS transmitter. Once all sources that wish to submit packets have been attended to, the new timeout is computed for the handler function. If all packets from a source have been transmitted, the source will be declared inactive, and will not be considered for future transmissions. The main loop will then be stalled until once again it is interrupted by the timer handler, in which case the process is repeated.

C.2.1 Implementation of the AAL2 Generator Timeout Mechanism

It was decided to use an alternative timer mechanism for the AAL2 generator as opposed to the one that was discussed in chapter 5, for two reasons:

- Only a single timer-CU is required, as the client need only support a single path, and is not thread aware.
- As the generator must emulate the effect of multiple sources, a timer for each source is required to ensure that the correct packet inter-arrival times are adhered to.

Source/Timer	Time (in base units)
Source A	3
Source B	7
Source C	5
Timer-CU	10

Table C.1: Source Inter-arrival Times

Reference Time	Source A	Source B	Source C
1			
2			
3	*		
4			
5			*
6	*		
7		*	
8			
9	*		
10			*
11			
12	*		
13			
14		*	
15	*		*
16			

Table C.2: Source Inter-arrival Times

To illustrate the operation of the timer mechanism, an example will be used. Assume the client must generate traffic for three sources, Source A, B and C, as well as a path timer, whose inter-arrival times and expiry time, respectively, are shown in Table C.1 above.

Now, if source A were to transmit 5 packets, source B, 2 packets and Source C, 3 packets, the packets would occur at the times indicated by (*) in Table C.2 above.

The fundamental task of the generator is to calculate the time the system should wait before the *next* interrupt should occur. The time is known as the critical time. Before the main loop has been entered, no packets have yet been transmitted, and therefore the timer-CU won't be running. As no time has yet elapsed, the reference time is now zero. To determine the critical time, a counter is commenced at the reference time (zero in this

case) and increments by one, until a critical time is found. With each iteration of the counter, a test is performed on the inter-arrival times of each source to determine whether any of these values will divide exactly into the counter value. As an example, when the counter reaches 1, neither 3, 7 or 5 can divide exactly into this number. Therefore, at this stage nothing is done, and the counter simply increments.. At 2, the same effect is noticed. At 3, it is found that the inter-arrival time of Source A divides exactly into this number. Hence the critical time is calculated as the difference between this time (3) and the reference time (0), and the counter need not increment any further.

Before setting the timer to 3 time units, an additional check is made to ensure this time does not exceed that of a possibly running timer-CU. For example, if the critical time is set to 3 time units, but a timer-CU would occur at 2 time units, then the critical time will be altered to 2 time units instead, as the timer-CU would have to occur first. When the timer-CU does eventually interrupt the system, the critical time will be recalculated, and the reference time will be updated. In this instance, the following calculated critical time would be 1.

More commonly, the critical time would be less than the current running timer-CU. Therefore, in the example above, a timeout would occur at 3 time units. When this happens, a check is made once again to see what sources are affected. If a source is affected, it is required at this stage to transmit a packet, provided it is still active. Once the full list of sources has been scanned, the new critical time is calculated. As the next timeout should occur at 5 time units, and since the reference time is 3 time units, the new critical time would be 2 time units. This process is repeated until all sources are no longer idle, at which point the generator will terminate.

C.2.2 Emulating Multiple Input AAL2 Paths

As the generator was implemented to multiplex data onto a single outgoing path, an entirely new instance of the algorithm, in its own process space, must be run for each additional output path that is required. Although the algorithms may be run on the same machine, each instance should be in a separate directory, with its own source definition file. One concern that manifested itself was that it was initially not possible to start all transmitting processes at the same instance in time. However, a simple wait instruction was inserted at the start of the generator code. When the generator is instantiated, it will only begin to stream packets into the AAL2 transmitter when the following calendar half-minute comes

around. Therefore, if a generator as started at a system clock time of 22:30:39, it will only run when a time of 22:31:00 is reached. This gives a user ample time to start additional generator instances, and works reasonably well for a small number of AAL2 paths.

C.3 The AAL2 Analyser

The AAL2 analyser is responsible for receiving AAL2 cells from the network and extracting the packets within each AAL2 cell. The architecture of the analyser is considerably simpler than that of the generator. No timer mechanism is required, packets need not be generated (as they already exist), and there is no need to parse a source file.

The operation of the analyser is as follows:

- Before the analyser can receive traffic from the network, an input path is created on a specified VC to receive the traffic, and the CPS receiver is initialised.
- For each AAL2 cell that is received, the PDU itself is logged to file, and the payload contents are submitted to the CPS receiver for processing. The receiver will consequently log all packets that are extracted to disk. A separate file is held for each group of packets with identical CIDs.

The analyser does not accept traffic on multiple AAL2 paths. To enable this feature, multiple instances of the analyser must be run, each as a separate process, and bound to a unique ATM VC. As no processing can be performed until cells are received from the network, it does not matter if the various traffic analysers are started at slightly different instances in time.

Appendix D

Abstract Data Type

D.1 Introduction

This appendix will give an overview of three Abstract Data Types (ADTs) that were implemented and extensively used for the AAL2 switch implementation. The notion of an ADT implies a generalised structure or code segment that, although is required by the particular switch implementation, may be re-used for any architecture that would require such an ADT. Furthermore, it was decided to make use of such ADTs for the following reasons:

- **Simplified interface:** The design of any ADT, whether a FIFO, linklist or hash table, requires a simple-to-use and standard interface. In this manner, one need not possess the technical know-how of a particular ADT in order to use it, and need only learn a simplified set of function calls.
- **Portability and re-use:** By coding each ADT in ANSI C, a higher degree of portability can be maintained. Although such code was originally intended to run in the NetBSD environment, it may easily be compiled on the respective platform of whatever future application is to be supported.
- **Simplified testing and robustness:** By first isolating the code for a given ADT before incorporating it into the main implementation, an extensive series of tests may be conducted, and the process of diagnosing faults is significantly simplified.

D.2 Generalised FIFO algorithm

Due to the requirement for buffers to be situated at multiple locations in the AAL2 switch, a series of queue managers were required. It was decided to base these queue managers on a simple FIFO discipline. A FIFO (First In First Out) simply implies that elements will leave the queue, i.e. be serviced, in the order in which they arrived, hence providing a certain degree of fairness¹. A special requirement of this FIFO algorithm, however, was that it would not be limited to the storage of only a single type of element, but would indeed be able to accommodate various element types. As an example, both CPS packets and AAL2 PDUs may require queueing in the switch. Both these structures are different in length and format, and may both be regarded as complex data types, whereas an unsigned character would be regarded as a simple data type². The FIFO should therefore support both simple and complex data types.

The feature of supporting multiple user-defined types is made possible by defining an array of void³ pointers. Before being placed in the queue, the size of the element to be stored must be determined and passed with the element itself. This is necessary in order to correctly delineate elements from one another in the queue, and to ensure the right number of bytes are copied for each element⁴. When retrieving elements from the FIFO (*dequeue* operation), a void pointer to a raw block of data in memory is returned. This pointer must first be type-casted to the appropriate type (e.g. *struct type_cps_packet* or *unsigned char*) before it may be used.

The code-base for the FIFO ADT may be found in *fifo.c*, and contains the following function calls:

D.2.1 Functions for creating and destroying FIFOs

- **Initialisation**

Before the FIFO algorithm may be used, it must first be initialised. The maximum

¹Note that there are multiple definitions of fairness in the literature, particularly that pertaining to scheduling mechanisms.

²Here the author defines a complex data type as one that may be broken up into simpler individual segments, e.g. a structure consisting of a 64-byte array of characters, preceded by 3 separate unsigned characters for header identification and classification purposes

³Not type-casted in any way

⁴Copying of element data onto the FIFO is performed in a raw fashion, byte for byte, without regard for the type of the element, until the specified element size is satisfied.

number of concurrent FIFOs that will be needed is the only parameter required.

```
int init_fifo (int NumFifos);
```

- **Creating a FIFO**

For each new FIFO required, *create_fifo* must be called with the relevant parameters. This will ensure that the FIFO's internal variables are set to appropriate values. These variables indicate, amongst other things, where free space exists to enqueue the next element, what elements must be dequeued next, the number of elements on the queue, the supported element size and how the internal structure of the FIFO should be organised. Parameters that are to be passed include the maximum number of elements to be supported by the queue, as well as the element size in bytes⁵. The return value from this function call is a descriptor that is to be used for subsequent access to the FIFO.

```
int create_fifo (unsigned char element_num,size_t element_size);
```

- **Destroying a FIFO**

When a FIFO is no longer required, it may be destroyed by passing its matching descriptor (obtained from *create_fifo*) to the function call below. The task of this function call is to free all memory occupied by the FIFO and mark the FIFO as unallocated. An unallocated FIFO may be assigned to the caller in the following instance of *create_fifo*, hence providing a primitive re-use mechanism⁶.

```
int destroy_fifo (int fifo);
```

D.2.2 Standard Element Processing

- **Enqueue**

Enqueue is the process of placing a new element on the FIFO, to be extracted at a later stage. Before an element can be placed on a FIFO, the descriptor of the

⁵Note that one limitation of the FIFO algorithm is that all elements common to a FIFO must be of the same data type.

⁶This mechanism was found to be adequate for the needs of the AAL2 switch, and ought to match those of many buffer-oriented applications.

FIFO must be identified, and the size of the element **must** match the size passed to the `create_fifo` call routine. If this is not adhered to, unpredictable behaviour and memory corruption will occur. This descriptor, together with a pointer pointing to the start of the element's memory block are passed together to the routine below. If successful, the value 0 is returned. If unsuccessful, e.g. the FIFO is full, a value of -1 is returned.

```
int enqueue (int fifo,void * element);
```

- **Dequeue**

Dequeue is the process of retrieving an element from the queue. Before an element may be retrieved, a storage area of the correct size and type must be preallocated. The correct FIFO descriptor, together with a pointer to this storage area are passed together. Provided the FIFO is not empty (in which case of -1 will be returned) the contents of the new element will be copied to the allocated storage area. When successful, a value of 0 is returned.

```
int dequeue (int fifo,void * element);
```

D.2.3 Miscellaneous Options (non-mandatory)

- **Number of Elements**

The number of elements held by a particular FIFO may be retrieved by passing the FIFO descriptor to the function call below:

```
int Fifo_Elements (int fifo);
```

- **Testing for FIFO Full or Empty**

2 additional function calls are provided to determine whether a FIFO is full or empty. A value of 1 (TRUE) is returned if the required condition is satisfied, and 0 (FALSE) for the opposite condition:

```
int IsFull (int fifo);  
int IsEmpty (int fifo);
```


D.3 Generalised Linklist Algorithm

Although many features of the FIFO can be found in that of a linklist, the two ADTs are not alike. Indeed, a linklist allows for insertion or extraction at either the head or tail of a linklist, unlike a FIFO, which must follow a strict access control discipline. Furthermore, in the FIFO algorithm, it is not possible to examine the contents of any other element without first extracting the element(s) from the front of the queue⁷, whereas elements of a linklist may be examined without altering the structure and order of the linklist.

The need for a linklist ADT arose from two separate requirements in the switch. Firstly, an individual linklist is required for each scheduling class in the PSU of the switch, as explained in chapter 5. Secondly, for each outgoing AAL2 path, a separate linklist must exist for the channel set that will ultimately indicate what channels will be combined at the path output. The linklist ADT code has been further extended to provide a rich set of additional features, to be explained shortly.

The multiple-type feature⁷ of the FIFO above has been implemented in a similar fashion in the linklist, and the same requirements exist for adding and removing elements, with regards to allocating sufficient storage, and making use of the correct linklist descriptor when accessing the various function calls. Furthermore, each individual linklist must also contain elements of the same type and size. Therefore once a particular data type has been specified, the linklist cannot accommodate another type unless it is destroyed.

The linklist implementation can be found in *linklist.c*, of which the various functional calls contained within are discussed below.

D.3.1 Creating and Destroying Linklists

- **Initialisation**

Before any linklists may be created and used, the algorithm must first be initialised. The only parameter required is the maximum number of linklists that need to be supported.

```
int init_linklist (int NumLinklists);
```

⁷The author does not see any direct need for such an operation to occur.

- **Linklist Create**

When creating a linklist, the size of the element to be stored must be known, and is the only parameter required. Note that unlike the process of creating a FIFO, there is no need to specify the number of elements that are to be supported in a linklist. This is because memory allocation of linklist elements is fully dynamic, and the only limitation on linklist size is the total memory capacity of the process. It was felt that a dynamic allocation scheme was more suitable in this case, as the linklist structure and size changes less frequently. The routine below will return a unique linklist descriptor, that is to be used for all further accesses to the linklist.

```
int CreateLinkList (size_t element_size);
```

- **Linklist Destroy**

When a linklist is no longer required, it may be destroyed, and the memory can be freed for another application.

```
int DestroyLinkList (u_int16_t linklist);
```

D.3.2 Standard Linklist Operations

- **Adding an Element to the Head**

Adding an element to the head of the linklist is simple. One need only possess the linklist identifier, as well as a pointer to the storage block in memory of the element, before calling the routine below:

```
int AddToHead (u_int16_t linklist, void * element);
```

- **Adding an Element to the Tail**

The format for adding an element to the tail is identical to the above scenario⁸:

```
int AddToTail (u_int16_t linklist, void * element);
```

⁸Note that when adding an element to an empty linklist, *AddToHead* and *AddToTail* will have the same effect.

- **Removing an Element from the Head or Tail**

If neither the head or tail elements are required, the appropriate function call may be used. If successful, a value of 0 is returned. If no elements can be extracted from the linklist, a value of -1 is returned.

```
int RemoveHead (u_int16_t linklist);  
int RemoveTail (u_int16_t linklist);
```

- **Get Head and Tail Elements**

The two function calls below may be used to non-obtrusively extract an element from a linklist⁹. The structure of the linklist will not be altered. Once the pointer to a block of memory is received from the calling routine, it must be type-casted to be usable.

```
void * GetHeadElement (u_int16_t linklist);  
void * GetTailElement (u_int16_t linklist);
```

D.3.3 Extended Operations

- **Get Element based on Index**

Occasionally, the need arises to linearly traverse a linklist's elements and read their contents. This is the case, for example, in the output channel set of an AAL2 path. A loop is initialised from 1 to the number of the elements of the linklist. In this manner, the current loop index may be passed to the routine below to retrieve the element at the specified location:

```
void * GetElement (u_int16_t linklist, int index);
```

- **Remove Element based on Index**

The process of removing an element based on an index is identical to the above scenario. A value of -1 is returned if the linklist is empty.

```
int RemoveEntry (u_int16_t linklist, int index);
```

⁹For a linklist containing a single element, both functions will return the same element.

- **Get Number of Elements**

The number of elements of a linklist is required when performing the above linear traversal, for example.

```
int GetNumElements (u_int16_t linklist);
```

D.4 Inverse Priority Heap

A priority heap is a structure similar to a queue, with the exception that elements are retrieved based on priority, e.g. the element with the highest value is retrieved first. An inverse priority heap implies that the lowest element is now given the highest priority. An inverse priority heap (IPH) was required to support multiple timers in NetBSD. The lowest time is always considered to be most critical, and hence must be extracted first.

The IPH is based on a binary tree, where heap order is satisfied. For heap order to occur, a parent cannot have a value lower than either of its children. For example, if a parent has the value of 7, and its children have values of 4 and 6, then heap order is satisfied. However, suppose that the children were instead 4 and 8. In such a case, heap order would not be satisfied, as 7 is less than 8. However, provided this constraint is met, the children need not be ordered from left to right, as is the case with a complete binary tree. Hence the children could have been 6 and 4, or 8 and 4, for case 1 and case 2, respectively.

Instead of storing an array of simple decimal values, the priority heap has been modified to store a structure, shown as follows:

```
typedef struct {
    char target_count;
    unsigned long target [10];
    unsigned long target_time;
} entry_type;
```

The *target_time* is the value to be used by the IHP algorithm. Hence, the structure to be retrieved from the IPH will always be the one with the lowest target time. *Target_count* refers to the number of targets to be found in the target array. The target array represents all sources that must be notified when a timeout occurs. For example, if at time $t=3$,

Source 1 generates a timeout of 4 time units, and at time $t=5$, Source 2 generates at timeout of 2 units, then at time $t=7$, the timeouts of both Source 1 and Source 2 will expire, and hence both sources will be notified.

The code for the IPH can be found in *priority_heap.c* and *pribrity_heap.h*.

D.4.1 Initialisation

The only routine required for initialisation is shown below. The routine will ensure an array is created¹⁰ to store the binary tree with heap order as explained above.

```
int InitPH (void);
```

D.4.2 Standard Operation

- **Inserting an Element**

Before an element can be inserted, the element structure must first be created and instantiated. A pointer of type *entry_type* pointing directly to this structure is passed to the routine below. The routine will examine the target time of the structure, and reorder the tree recursively until heap order is satisfied. This allows for a simpler remove algorithm that operates substantially faster¹¹.

```
int InsertPHElement (entry_type * new_heap_entry);
```

- **Removing an Element**

Once again, the element with the smallest target time will be removed. The structure comprising this element will be copied to a preallocated area of memory, pointed to by argument *theElement*.

```
int RemovePHElement (entry_type * theElement);
```

¹⁰The array is limited to 30 elements. It was not envisioned that more than 30 unique timeout values would be required. However, this limit is easily extended.

¹¹Reordering of a tree can be expensive with many elements.

Appendix E

WUGS, APIC and SPC Usage Guide

E.1 Introduction

This appendix describes the physical procedures to be taken out to configure the WUGS environment to support a given architecture, such as an AAL2 switching module. This appendix may therefore be used as a quick-start guide for those unaware of the WUGS operating environment, and who wish to get the AAL2 switch “up and running” as soon as possible. The following prior technical knowledge will prove helpful:

- A good understanding on the general layout of the WUGS, AAL2 switching module and testbed components
- A good understanding of NetBSD (or any BSD-based environment) and the ability to recompile a kernel

E.2 Setting up of the WUGS Controller

The only way to configure the VXT table of the WUGS is via an external controller that transmits raw ATM cells on VCI 32. Therefore the first step in constructing an operational WUGS environment is to setup and configure the WUGS controller. The WUGS controller is simply an end-station (PC) equipped with an APIC NIC to transport the raw ATM cells. To install the machine, the following should be done:

- A version of NetBSD 1.4.1 must be obtained.
- Following the INSTALL directions, NetBSD should be installed on the WUGS controller. This is a relatively straightforward process, and won't be explained here. Once the installation is complete, the following additional non-WUGS-related software is also recommended:
 - GNU Bash, a user-friendly interactive shell
 - Secure Shell, preferably OpenSSH, to allow users to remotely access the machine in a secure fashion¹).
 - Vim, or Vi Improved, a text editor that support syntax highlighting².
- The Washington University SPC MSR (ex-Crossbow) distribution must be obtained via their cvs server, *cvs.arl.wustl.edu*, and should be extracted in the */usr/downloads* directory of the newly installed machine. This distribution includes a full set of GNU development tools, modified APIC kernel, full SPC operating environment and all WUGS utilities required.
- Once extracted, the directions in the README file in the root directory of the distribution will guide one on how to complete the installation of the above mentioned utilities, and how to compile the new APIC-aware kernel. Once the kernel has been installed, the machine will need to be rebooted, and checked with *"dmesg / less"* to test whether the kernel has detected the new APIC NIC.
- The installation of the WUGS controller is now complete.

E.3 Running GBNSC and Jammer on the WUGS Controller

To enable the controller to communicate with the WUGS switch, the following requirements must be met:

- A port on the WUGS must be directly connected to the APIC NIC of the host-based controller. In the example below, port 4 was used as the control port.

¹OpenSSH will require OpenSSL and Perl to be installed on the machine beforehand.

²Vim required ncurses to be installed on the machine beforehand.

- Before control cells may be sent to the WUGS, a PVC to the WUGS must be set up with a VPI/VCI of 32 (decimal). This is achievable with the following commands:

```
/sbin/ifconfig apic0 up
/usr/sbin/atm_ifconfig apic0 0x000020 open -inport FIBER -outports FIBER -aal0_cell
-nolc ratm
```

- To create the control cells that will ultimately be sent to the WUGS, a program known as the Gigabit Network Switch Controller (GBNSC) must be running on the controller. GBNSC will listen for commands over a TCP socket from Jammer (discussed below) and convert these commands to raw ATM cells that can be interpreted by the WUGS. A complete overview of GBNSC can be found at the following URL: GBNSC Overview http://www.ststech.com/.../Switch_Software/The_GigaBit_Network_Switch_Controller,_Version_1.3.pdf. When the GBNSC process is initialised, a configuration file is read in that describes the manner in which the WUGS is connected to the controller (port number and control VC) as well as flow control information (UNI vs NNI ports). An example script can be found in Appendix I. In order to initialise GBNSC, the following command should be issued, directly from the directory where itself as well as the matching configuration file may be found:

```
GBNSC.apic -a -b64000 config.apic
```

- At this stage, the controller is ready to receive commands from Jammer. Jammer may operate in two modes. The first mode is an interactive mode, where a user specifies Jammer-based commands directly. The second mode is a script-based mode, where Jammer will read a set of commands from a file. As the Jammer commands are relatively cryptic, and not intuitive, a hybrid mode was selected to create an interactive, user-friendly environment for configuring the VXT table. A script containing a set of standard function calls was released by the Georgia Institute of Technology, and is available on the main Washington University Gigabit Kits website. The following commands are provided:

The script to enable these commands is included in Appendix I, and can be incorporated with Jammer in the following manner:

Command	Function
Initialise ()	Initialise port maintenance registers
AddPvc2 (SourcePort, SourcePath, SourceChannel, DestPort, DestPath, DestChannel)	Create a bi-directional VC between source and destination ports
AddPvc (SourcePort, SourcePath, SourceChannel, DestPort, DestPath, DestChannel)	Create a uni-directional VC from source to destination port
AddPvc2 (SourcePort, SourcePath, SourceChannel, DestPort, DestPath, DestChannel)	Remove already-active bi-directional VC

Table E.1: Commands to establish and tear down WUGS connections

Jammer 0.1 *inca* 3552 kitsppp.js

“0.1” refers to cluster 0, switch 1. Jammer will connect to GBNSC on port 3552 (provided GBNSC is listening on this port) of host *inca*. The script file to be included will be called *kitsppp.js*. More information on the Jammer utility may be found at the following URL: Jammer URL <http://www.seas.smu.edu/~barr/wugs/jammer.pdf>.

It is also possible to control GBNSC from a remote end-station, which itself must run an instance of the Jammer utility. The difference here will be the host name, which will not be the name of the host itself. For example, if host *aztec* wishes to control the GBNSC utility on *inca*, the syntax of the Jammer command will be identical to the example above (and does not change to *aztec*).

E.4 Migrating an architecture to the SPC

Since the architecture and components of an end-station are almost identical to that of the SPC, it is a relatively simple process to migrate an implementation from the one platform to the other. The following procedure shows how this may be accomplished:

- As the SPC contains a ramdisk that will house the entire file hierarchy of the operating system, the first step that needs to be performed is to prepare the filesystem. A working filesystem is provided with the MSR or Crossbow distribution that would

have been installed above, and is given the name *SPC24MB.fs*. The file can be found in the SPC root directory, usually called *spc_working_dirs/root_wdir*, which is off the main root directory of the distribution. Note that as the filesystem is stored as a contiguous image, it must first be mounted before any modifications can be made. To select a suitable mountpoint, the script “*mountit*” should first be edited. Following this, the script may be run, which will first attach a device node to the image file, and then mount the file in the specified location in the normal way.

Now that the file system can be accessed directly, the development code may be copied to a suitable location, e.g. */usr/local/AAL2Switch*. If any external library files are required, e.g. *libpth*, these should be copied to the standard library directories, which are usually */lib* or */usr/lib*³. Once configuration of the filesystem is complete, it may be unmounted using the “*unmountit*” script⁴.

- Before the filesystem can be downloaded, the SPC kernel must first be rebuilt. A symbol-only version of the kernel will be added to the root of the filesystem and the filesystem and kernel will be concatenated into a single image before being downloaded. Before the kernel is compiled, it should first be configured by editing the appropriate configuration file. As an example, the crossbow kernel configuration can be found at the following location relative to the root of the distribution: *crossbow/NetBSD/usr/src/sys/arch/i386/conf/SPC_24_MB_with_Crossbow*. Configuration of the kernel should be performed by first changing to the correct directory before editing the file. The user may then proceed by running *config SPC_24_MB_with_Crossbow* to prepare the build environment. The build scripts can be found in the directory *../compile/SPC_24_MB_with_Crossbow*. Once in this directory, the user should run *../mk.1*, a script provided by Washington University, to build the kernel image and copy this image to the root spc working directory.
- By returning to the root SPC working directory, the symbol-only version of the kernel can be extracted and added to the SPC filesystem⁵. This is performed by running the *mk.2* script in the current directory.
- The user now returns to the directory *crossbow/NetBSD/usr/src/sys/arch/i386/compile*

³Warning: A common trap to fall into is to forget that the SPC filesystem root actually falls under the mountpoint specified, e.g. */mnt/spc*. A user should be careful to ensure that their files are not copied to main filesystem of the workstation.

⁴The *mountit* and *unmountit* scripts must be carried out by the superuser.

⁵Also to be performed by the superuser.

to run the *mk.3* script. This script will append the kernel image to the SPC filesystem, and create a *bss* file that is suitable for downloading to the SPC.

- Before the entire image may be downloaded to the SPC, two PVCs must be opened to the SPC. The SPC is assumed to reside on the same switch port to which the end-station is connected, that will be responsible for downloading this file. The first PVC is an AAL5 connection to the SPC that will be used for downloading the frames that represent the image. An AAL0 connection is also initialised to allow the SPC to provide feedback to the end-station. The various required connections may be configured by running the *doit.sh* script in the *spc_working_dirs/configuration* directory, off the main distribution root directory.
- The image may now be downloaded to the SPC. First, the user should change to the *spc_working_dirs/downloads* directory off the root of the main distribution. Before downloading may commence, the SPC must first be reset. This is performed by running the "*resetit.sh*" script, which will send a reset control cell to the SPC. Once complete, the image comprising the filesystem and kernel may be downloaded by running the "*doit.sh*" script.
- The downloads process will take approximately four minutes. While waiting, the user should open a serial console to the SPC which will be used for shell access.
- Once the download is complete, the user may log in to the SPC as root, configure the required APIC connections to the switch, and run the relevant software. If an error in the filesystem is detected, or required code or libraries are missing, or if an additional script or tool is provided, the entire downloading process must be repeated. As this consumes considerable time, a suggestion is to make use of the NFS protocol to mount a remote filesystem. This has the following advantages:
 - Any changes that are made will not be lost when the SPC is reset or the WUGS loses power.
 - Code can be edited and recompiled on the end-station, which is equipped with the necessary development libraries, headers and compilers. The resulting binary file can then be directly run on the SPC to test its behaviour. It would not be feasible to place an entire development library on the SPC, as the filesystem is only 24 MBytes in size.

Appendix F

AAL2 Switch and Client Code Overview

F.1 Introduction

The AAL2 switch, generator and analyser projects combine multiple source files, each of these files accomplishing a small task. To address certain portability issues and to maximise the re-use of already existing code, certain files appear in both the switch and/or both end-station environments. This appendix will name all these code components, with a description of each.

F.2 AAL2 Switch Code Environment

The code comprising the core AAL2 switch module is made up of the following components:

- “aal2.h”
Main AAL2 header file. Contains structures to hold an APIC cell, a CPS Packet, an AAL2 PDU, the transmitter state machine structure, and the receiver state machine structure.
- “adu.c”
ADU Module. This module reads in AAL2 PDUs and passes them to the receiver for processing. Once complete, control is relinquished.

- “aru.c”
ARU Module. This module removes CPS packets from the output stores and passes them to the AAL2 transmitter. Once all possible packets are extracted, control is relinquished.
- “ccu.c”
CCU Module. Performs initial configuration of the system, and starts additional threads. The main function, i.e. the first function to receive execution can be found here.
- “fifo.c”
Abstract FIFO-manipulation routines.
- “linklist.c”
Abstract linklist-manipulation routines.
- “priority_heap.h”
Abstract priority heap data structure header file.
- “priority_heap.c”
Abstract priority heap data manipulation routines.
- “psu.c”
Packet Scheduling Unit. Initialises low-delay, normal-delay and high-delay linklists. Passes channels from input to output store according to scheduling class, and relinquishes control.
- “receive.c”
CPS Receiver. Contains state machine driven by input AAL2 PDUs. Non-signalling packets are copied to the input store. Signalling packets are copied to the dedicated signalling queue set aside. Once the PDU has been fully analysed, and provided any signalling packets were extracted, an event is generated to the SPU. Once complete, control is relinquished.
- “spu.c”
Signalling Processing Unit. Contains routines to start instances of an ADU or ARU, assign channels to input paths, append channels to an output channel set, create input AAL2 paths, create output AAL2 paths as well as a state machine to process

incoming signalling packets. Once all packets have been processed, control is handed over to the main scheduling unit.

- “switchcore.h”
Main AAL2 switch data structure header file. Contains signalling event identifiers and templates for a table mapping structure, an output channel set structure, input and output AAL2 path structures and a structure to hold signalling information.
- “tools.c”
Miscellaneous tools. Contains utilities to subtract times for analysis purposes, functions to convert CPS packet headers from serialised form to individual identifier form and vice versa, and functions to convert the start field of a PDU from serialised form to individual identifier form.
- “transmit.c”
CPS Transmitter. Contains state machine to construct AAL2 PDUs, and pass complete PDUs to the output link.
- “tu.c”
Timer Unit. Contains routines to set and reset a new timer, a timer handler function to catch timer interrupts, an intelligent timer wrapper to make use of a single system timer to create the illusion of multiple timers, and a routine to cancel already active timers, for when PDUs becoming full and longer require the timer.

F.3 The AAL2 Generator

The generator contains the following code components:

- “aal2.h”
Main AAL2 structure header file.
- “aal2term.h”
Terminal header files. Contains similar structures as found in aal2core.h, but modified for end-terminals.

- “client.c”
Main analyser / client source file. Contains timer functions and wrappers, as discussed in Appendix C, and a main loop to extract packets from memory at the correct times and pass them to the CPS transmitter.
- “excel.c”
External program (not compiled in with analyser) to convert logs produced by the analyser into a column-aligned format, suitable for reading by MS-Excel¹.
- “file.c”
Routines for reading packets and PDUs into memory. This is required to ensure that data for the client may be written directly out of memory, instead of reading from a file multiple times, slowing down the entire process.
- “packetgen.c”
Packet generation routine. Generates packets with specification as defined in sources.conf (see Appendix C), and utilises the math library random routine to produce random payload data.
- “parser.c”
Parser routine to read specifications from source.conf file. Converts identifiers to tokens, matches tokens and assigns parameters to data structures.
- “sources.h”
Source description header file. Specified source template to fully describe the attributes of a source.
- “sources.c”
Write various generator events to logfile, that will later be processed by the excel function.
- “sources.conf”
External source configuration file.
- “tools.c”
Miscellaneous functions for converting between various packet header and start field formats.

¹The excel routine was used as a preprocessor for the graphs of chapter 6.

- “transmit.c”
CPS Transmitter.

F.4 The AAL2 Analyser / Receiver

The analyser contains few components. They are:

- “aal2.h”
Main AAL2 structure header file.
- “aal2term.h”
AAL2 terminal structure header file.
- “file.c”
Routines for reading packets and PDUs from file into memory.
- “receive.c”
CPS Receiver.
- “server.c”
Main server file. Reads AAL2 PDUs from the networks, submits data to the CPS receiver and logs all activity.

Appendix G

Accompanying CD-ROM

This text is accompanied by a CD-ROM. Before navigating the CD-ROM, the user should first read the README file in the root directory, which will contain any last minute information. The CD-ROM will contain the following information:

- **Source Code**

All source code that was produced can be found in the "Sources" directory.

- **Source Code Documentation**

Documentation that proved invaluable, and to which the author is extremely grateful, can be found in the "Source-docs" directory

- **WUGS Environment**

All WUGS-related sources and utilities that were used for this thesis can be found in the "WUGS" directory.

- **WUGS Documentation**

Documentation describing the WUGS, SPC and APIC can be found in the "WUGS-docs" directory.

- **Conference Proceeding and Journal Articles**

Electronic copies of research papers, some which are listed in the "References" section of this text, can be found in the "Research Literature" directory.

- **Thesis document**

This document, in both postscript form and pdf form, can be found in the "Manuscript" directory.