A Robust, Efficient Physical Layer Transport Protocol for Packets

by

Paul Langner, B.Eng., M.Eng., P.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Department of Electronics

Carleton University
Ottawa, Ontario
September, 1999

0-612-48347-9

Carleton
UNIVERSITY

The undersigned recommend to the Faculty of Graduate Studies and Research acceptance of the thesis,

"A Robust, Efficient Physical Layer Transport Protocol for Packets"

submitted by

Paul Langner, B.Eng., M.Eng., P.Eng.

in partial fulfilment of the requirements for the degree of Doctor of Philosophy, Engineering.

[illegible]

Chair, Department of Electronics

[illegible]

Thesis Co-supervisor

[illegible]

Thesis Co-supervisor

[illegible]

External Examiner

Carleton University
September, 1999

Carleton UNIVERSITY

Abstract

While the majority of data transmitted in today's telecommunications networks is packet oriented, there is no robust physical layer framing mechanism designed specifically for packet data. As a result, the packet data must be mapped either into ATM cells and then into a SONET/SDH frame, or encapsulated in PPP, HDLC, and then into SONET/SDH.

In this thesis, the various methods of format synchronization are explored in the context of packet transmission. A novel, packet oriented physical layer framing protocol based on CRC-16 packet boundary delineation is developed that is both robust and efficient. This protocol is then analyzed from a performance analysis perspective. Finally the physical implementation of an integrated circuit containing transmit/receive framers using this protocol and its performance is examined.

Carleton
UNIVERSITY

Acknowledgements

I wish to express my gratitude to my thesis supervisors, Ralph Mason and Martin Snelgrove (thanks Martin for putting up with me all those years), and to Northern Telecom, Fujitsu, and Lucent for funding me to do my Ph.D. I would also like to thank Subra Dravida, Enrique Valencia-Hernandez, and James Manchester for valuable discussions on the SDL framing and scrambling structure, and Ed Roberts for his unwavering support of SDL. Finally I would like to thank my parents, my father-in-law, and my wife for their support and encouragement over the years.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Figures

Carleton UNIVERSITY

List of Tables

List of Tables

Table | Page

List of Symbols

Symbol	Definition
α	The number of events required to drop out of frame sync in an $\alpha - \delta$ framer.
α	A primitive root of a Galois field polynomial.
δ	The number of events required to acquire frame sync in an $\alpha - \delta$ framer.
β	The probability of a logic one occurring in an incoming binary data stream.
$\vec{c}$, $c(x)$	A codeword.
$d_{\min}$	The minimum distance of a code.
f	Normalized frequency.
$g(x)$	A generator polynomial for a Galois field code.
i, j, k, m, n	General counting variables
$\vec{m}$	A message vector.
$\vec{n}$	The current state of a transition matrix.
p	The probability of an event occurring.
p_{nm}	The probability of transitioning from node m to node n in a state transition diagram.
$p(n, m)$	The probability of a random binary vector of length n having m or fewer zeroes or ones.
$p(n, 2t)$	The probability of a random binary vector of length n having $2t$ or fewer transitions.
$p(x)$	A Galois field primitive polynomial.
$\vec{r}$	The remainder after Galois field division.
x	A finite field variable.
B	A framer in a bad presync state.
B_n	The nth Bernoulli number.
BER	Bit Error Rate.
D	A delay operator equivalent to one unit time interval.
D_n	The nth flip-flop.
$\vec{D}$	A vector representing the state of all of the flip-flops in

List of Symbols

Symbol	Definition
	a circuit.
$D(n,m)$	The number of binary vectors of length n having m or fewer zeroes or ones.
F	A framer in the false frame state.
G	A framer in a good presync state.
$\vec{G}$	A generator polynomial for a Galois field code.
$GF(q^m)$	The Galois field constructed using m-dimensional q-ary vectors.
H	A framer in the hunt state.
$\vec{I}$	The identity matrix.
L	The number of ones in a period of a self-synchronizing scrambler's impulse response.
L_n	The nth bit in the length field of an SDL header.
L_{AVG}	The average number of bits in a packet.
L_{CRC}	The number of bits in a CRC
L_F	The number of bits between frame patterns.
L_P	The number of bits in a frame pattern.
L_{Marker}	The number of bits in a frame marker.
L_{Max}	The maximum number of bits in a packet.
$M(x)$	A minimal Galois field polynomial.
P	The number of bits in a self-synchronizing scrambler's impulse response.
P_{XY}	The probability of a framer transitioning from frame state Y to frame state X, where the frame state may be any one of H, B, G, F, S.
P_x	The probability of event x occurring.
S	A framer in the sync state.
$S(f)$	Power spectral density.
T	Normalized period.
T_{nm}	The multiplicative weight associated with an edge

List of Symbols

Symbol	Definition
	between the start node m and the destination node n in a state transition diagram.
$T(n, t)$	The number of binary vectors of length n having t or fewer transitions when repeated ad infinitum.
$\vec{T}$	A transition matrix.
$\vec{X}$	A Galois field magic number.

Glossary

Term	Definition
AAL	ATM Adaptation Layer - the logical layer responsible for SAR operations for ATM.
ACK	Acknowledgement - the most common is a 40 byte (20 TCP, 20 IP) packet used in the TCP protocol to ensure guaranteed delivery of TCP/IP datagrams whose bytes are sequence numbered.
ADM	Add Drop Multiplexer - a piece of SONET/SDH equipment designed to interface to a protected ring carrying multiple channels, some of which pass through, and others which are dropped.
ANSI	American National Standards Institute - a North American standards body, similar to ITU.
ATM	Asynchronous Transfer Mode - a packet switched format employing 53 byte packets called cells, which have 5 bytes of header information, and 48 bytes of payload.
Backbone	A term used to describe the SONET/SDH ring based telecommunications network running at speeds typically greater than or equal to 155 Mb/s.
Bellcore	Bell Communications Research - the industry standards consortium set up to provide network standards for North American telecommunications. Since the breakup of AT&T, its power is waning.
BER	Bit Error Rate.
BIP	Bit Interleaved Parity - a parity check typically used in transmission systems to detect payload corruption.
BIST	Built in self test - typically refers to a circuit associated with embedded memory structures in integrated circuits that performs testing on the memory at power-up.
Call Setup	A term typically associated with connection oriented protocols (such as ATM) used to describe the initiation procedure involved in setting up the connection.

Glossary

Term	Definition
Circuit Switched	A term used to describe a (typically) constant bit rate digital connection that is similar to a "patch panel" connection. This is in contrast to a packet switched network.
Collision Based	This refers to a physical layer protocol that uses a non-arbitrated, shared medium and can detect and deal with collisions due to simultaneous transmission.
Connectionless Protocol	A term used to describe a communications protocol which does not have to maintain a constant connection between the source and destination (i.e. with keep-alives, etc.). An example is UDP.
Connection Oriented Protocol	A term used to describe a communications protocol which maintains a constant connection between the source and destination (i.e. with keep-alives, etc.) An example is TCP.
CPE	Customer Premises Equipment - privately owned telecommunications equipment used to build LANs.
CRC	Cyclic Redundancy Check - a field appended (typically) to a message based on cyclic codes to provide a statistical guarantee of the message's correctness.
CSU/DSU	Customer Service Unit / Data Service Unit - a box (it used to be 2 boxes) that provides a data connection over a leased line using proprietary mappings into the payload of the leased line - i.e. T1, etc.
Datagram	A data message, that is typically transmitted as the payload in UDP or TCP.
DARPA	Defense Advance Research Projects Agency - a Department of Defense agency focused on funding advanced research projects of military interest.
Ethernet	A collision-based, physical layer protocol for constructing LANs using either twisted pair (increasingly popular) or coaxial cable. Popular forms today are 10BaseT and 100BaseT which are 10 Mb/s and 100 Mb/s Ethernet over twisted pair, respectively.

Glossary

Term	Definition
ETSI	European Telecommunications Standards Institute - a European industry standards body similar to Bellcore.
Frame Relay	A popular circuit switched data transport protocol used for packet networks and intended for ATM internetworking.
HDLC	High-level Data Link Control - a simple framing method designed for packets.
HEC	Header Error Check - the CRC used in the header of ATM cells.
IETF	Internet Engineering Task Force - the standards body that controls the standards for the Internet, IP, TCP, etc.
IP, IPv4	Internet Protocol - a connection-less protocol used for packet delivery. See [5].
ITU	International Telecommunications Union - the international standards body governing telecommunications standards for SDH countries.
Keep-Alive	A facet of a connection oriented protocol which maintains the link status with timers, etc.
LAN	Local Area Network - a term used to describe a CPE network, as opposed to the WAN.
LFSR	Linear Feedback Shift Register - a shift register used to implement the hardware equivalent of long division in a finite field.
LSB	Least Significant Bit.
MAC	Media Access Control - the digital signalling layer associated with an Ethernet link.
MPOA	Multiprotocol Over ATM - an encapsulation method used to transport protocols such as TCP over ATM.
MPLS	Multiprotocol Label Switching - also referred to as tag switching (as pioneered by Cisco), this involves attaching a tag to the packet for ease of routing purposes (similar to a VCI/VPI in ATM).
MSB	Most Significant Bit.
MTTF	Mean Time To Frame.
MTTS	Mean Time To Synchronization.

Glossary

Term	Definition
OAM&P	Operation, Administration, Maintenance and Provisioning - this refers to the control scheme of a network.
OC-n	Optical Carrier - n. The SONET line rate corresponding to n x 51.84 Mb/s.
Packet Switched	A term used to describe a network which switches individual packets on the fly, based on the information contained within the packet.
PDH	Pleisiochronous Digital Hierarchy - the first set of digital carriers defined for the public data network. An example of this is a T1 connection.
PLF	Probability of Loss of Frame.
PFF	Probability of False Frame.
PFS	Probability of False Synchronization.
PPP	Point-to-Point Protocol - an encapsulation method for datagrams that was designed to handle dial-up connections and identify the protocol of the encapsulated datagram. See [15].
PSTN	Public Switched Telephony Network- the telecommunications network, sometimes referred to as the WAN.
Quality of Service	A term used to describe the parameters associated with a connection with regards to delay, maximum burst size, etc.. It is important when trying to simultaneously switch real-time services (such as voice and video) with data services.
RFC	Request for Comments - the final form of an IETF standard.
SAR	Segmentation and Reassembly.
scan	A test methodology for integrated circuits that involves connecting flip-flops together in a serial chain to allow values to be read and written, allowing the verification of the combinatorial logic within the integrated circuit.

Glossary

Term	Definition
SDH	Synchronous Digital Hierarchy - the ITU version of SONET.
SDL	Simple Data Link - the link-layer protocol that this thesis discusses.
SONET	Synchronous Optical Network - a standard developed by Bellcore in the late 1980s to allow for the transport of high-speed, circuit-switched data via optics. Common rates are based on multiples of 51.84 Mb/s and include 155 Mb/s (OC-3), 622 Mb/s (OC-12), 2488 Mb/s (OC-48), and 9952 Mb/s (OC-192).
STS	Synchronous Transport Stream - the 51.84 Mb/s (an STS-1) frame structure that forms the basis of SONET.
T1	A 1.544 Mb/s line which is one of the PDH rates.
TCP	Transmission Control Protocol - the secure delivery mechanism for IPv4 packets. See [6].
TDM	Time Division Multiplex - a multiplexing scheme using fixed time slots in a repeating fashion. An example of this is STS-1 channelization in SONET.
UDP	User Datagram Protocol - another delivery mechanism for IPv4 packets, typically used for non-critical data. See [4].
UTOPIA	Universal Test & Operations Interface for ATM - a popular physical layer interface for ATM cells. The Level 2 standard is the most common and is typically implemented as a 16 bit wide 50 MHz bus with handshake signals. The proposed Level 3 standard operates at 100 MHz with 32 bits.
UTOPIA POS-PHY	A defacto standard extension of UTOPIA to handle packets other than 53 bytes in length. This involves partial packet transfer (typically 64 bytes) and some additional handshake flags.
VCI	Virtual Circuit Identifier - a 16 bit field in an ATM cell header identifying the virtual circuit, which is a local connection identifier.

Glossary

Term	Definition
VPI	Virtual Path Identifier - an 8 or 16 bit field in an ATM cell header identifying the virtual path, which is a local connection identifier for a bundle of VCIs.
VT	Virtual Tributary - the sub-STS-1 rate frame structure used by SONET to transport PDH signals. The smallest increment is a VT1.5 which is roughly 1.5 Mb/s.
WAN	Wide Area Network - a datacom name for the PSTN.
X.25	An old ITU packet transport standard.

Introduction 1-

1.1 Background and Objective[†]

Since data transmission statistics for the Internet have been tracked (~1969), data traffic volume has been roughly doubling[1] every year. It is estimated that in 1996 - 1997, voice and data traffic in the backbone reached parity in North America, with the rest of the world roughly 1 - 2 years behind[2][‡]. This data is almost exclusively packet based[3], and according to current analysis is mainly TCP (transmission control protocol) or UDP (user datagram protocol) packets which implies IPv4 (internet protocol version 4) payloads[4][5][6].

Since the backbone is virtually all SONET/SDH (Synchronous Optical Network / Synchronous Digital Hierarchy)[7][8], the only officially sanctioned means of transporting packets within this network is to chop them up and transport them via ATM (asynchronous transfer mode) cells, which are then mapped into SONET/SDH using an approved mapping[9][11][*]. The issue with this is that if you are building a packet switched network, you are forced to build interfaces to the WAN (wide area network) that can talk ATM. While the argument is somewhat

[†] For those not versed in telecom technology, the background to this thesis is that a new, "efficient" standard for transporting packets (packet over SONET) has been developed which has several serious flaws.

[‡] This is in contrast to the voice traffic annual growth rate of 17%.

[*] Note that ITU, Bellcore, ETSI, ANSI, and the ATM Forum all standardize various aspects of SONET/SDH, and all standards are slightly different.

religious in nature, the ATM signalling stack is quite involved, and there are not yet solid, clean mappings from packet network signalling into ATM signalling[†]. The result is that some amount of proprietary implementation is required.

Another argument against ATM is that there is a "cell tax" or overhead associated with mapping packets into cells. The obvious one is the extra 5 overhead bytes per 48 bytes of data in ATM, which equates to 9.4% wastage of bandwidth. More subtle is the fact that TCP uses small (i.e. 40 byte) ACK packets (packets containing only acknowledgement information) which expand to 56 bytes when encapsulated using MPOA (multiprotocol over ATM), thus effectively using 2 cells (106 bytes) to transport 40 bytes[10]-[14]. Depending on the ratio of ACK packets to datagrams, the average efficiency of the network may be quite low. For instance, if for every long datagram, an ACK packet is required, a 35.8% wastage would result (the average of 9.4% and 62.3%).

Consequently, a new standard (RFC1619) was developed to allow the direct mapping of packets into a SONET/SDH payload[15][16]. Unfortunately, this mapping has several flaws, including variable packet size expansion, and susceptibility of loss of packet from a single bit error. This will be discussed in detail later.

[†] Especially in the quality of service and call setup area.

Finally, there is the issue of using SONET/SDH as the physical layer transport medium. It has very complex, proprietary signalling associated with it (especially in backbone ring applications), as well as a 4.4% overhead tax[†].

This thesis addresses all of these issues and proposes a new framing protocol based on CRC-16 (16 bit cyclical redundancy check) packet boundary delineation that will allow robust, efficient mapping of packets into SONET/SDH or directly over fiber.

1.2 Thesis Organization

Chapter 2 of this thesis discusses packet transmission technology, from a framing perspective. Chapter 3 explores the different methods of providing frame boundary delineation while Chapter 4 explores the different methods of providing scrambling. In Chapter 5, the design of the new protocol is presented, and in Chapter 6 the performance of this protocol is analyzed. In Chapter 7 the implementation of this protocol in a 0.25μ CMOS integrated circuit is discussed, while Chapter 8 presents the measurements of this framer in the lab. Finally Chapter 9 presents suggestions for future work in this field.

[†] It should be noted that this signalling scheme is independent of the actual SONET/SDH transport mechanism, and SONET/SDH remains perhaps the most efficient means of transporting TDM traffic ever implemented.

1.3 Statement of Contribution

Unless otherwise stated, all of the material dealing with CRC based packet boundary delineation is the result of original work performed by the author over the period 1997 - 1999. Specifically the major contributions are:

1) the development of a workable framing protocol from the initial concept of using length based frame delineation with CRC-10 tagging[†][18].

2) the development of the numerical techniques to analyze framing protocols for an arbitrary number of parallel framers.

3) the analysis of the framing protocol under various conditions.

4) the analysis of the flaws in the current POS (packet over SONET) standard.

5) the analysis of how to implement a bit-parallel distributed sample scramblers.

The significant secondary contributions are:

1) the design and coding of the framing protocol as two blocks in an integrated circuit (the TDAT042G5).

2) the design of the bit-parallel implementation of the 48th order set-reset scrambler.

[†] The original concept of combining the length of the packet with a CRC was invented in Bell-Labs and used a CRC-10. Unfortunately this scheme was unworkable due to many factors (discussed in Chapter 5).

3) The acceptance of SDL as a standard in the ITU in G.707[8], and within the IETF (Appendix J)[†].

4) the design of the scrambler synchronization and "A" and "B" OAM&P (operations, administration, maintenance, & provisioning) messaging channels.

5) the discussion on the taxonomy of scramblers.

6) The application for a patent on the implementation of the ATM x^{31} algorithm.

7) The application for a patent on the multichannel transmit flow control structure of the data framer section.

[†] It is anticipated that the SDL specification will become a standard in the IETF by the end of the year, as all that was required for full standardization was the granting of path signal labels (C2) for SDL by the ITU. This was done in the November ITU T1X1 meeting.

Packet Transmission Today 2-

2.1 Introduction

As was mentioned in the introduction. there are two popular methods of transporting packets in today's network: packet over ATM over SONET/SDH over fiber. and packet over PPP (point-to-point protocol) over HDLC (high-level data link control) over SONET/SDH over fiber. In this chapter. both of these methods. as well as packet over ATM over fiber. are explored and the framing associated with them examined.

2.2 Packets

The Internet Protocol (IP) originated from work funded by DARPA in the 1970's to construct a reliable computing network for the US Department of Defense. Associated with IP are two additional protocols: TCP and UDP. with TCP being intended for reliable packet delivery. and UDP being intended for more rapid. less reliable. packet delivery (i.e. delay sensitive traffic. like voice). Today. IPv4 in its two principal flavors - TCP and UDP - constitutes the bulk of the data traffic on the Internet. This growth is being driven from the LAN computing environment which is today dominated by TCP/IP transported over Ethernet.

Figures 2.1 and 2.2 illustrate the headers for both TCP and UDP. As can be seen. the difference is in the information in the TCP and UDP fields. Since TCP is

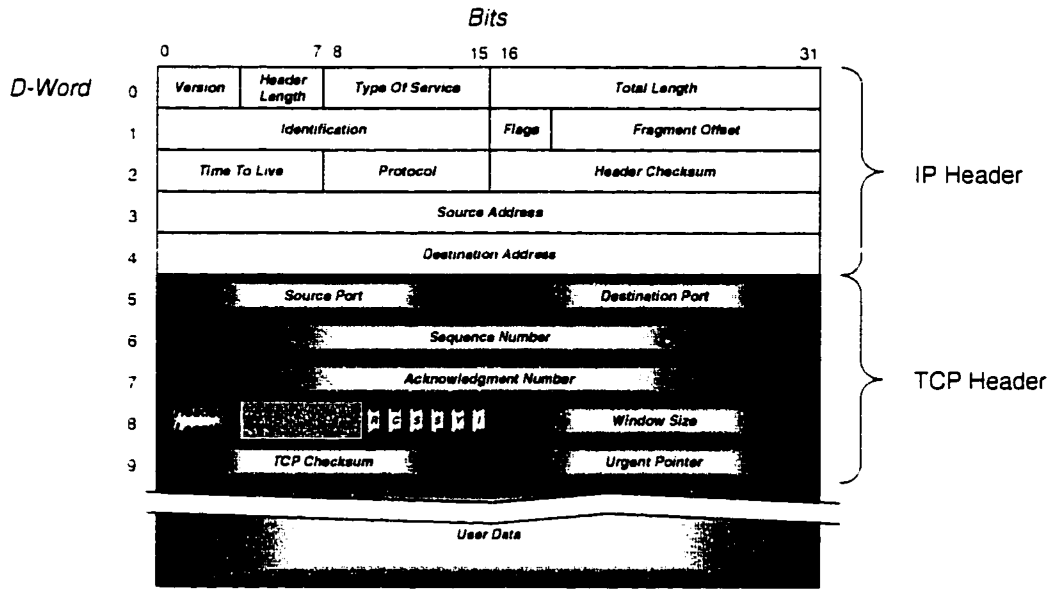


Figure 2.1 **TCP packet format with IPv4 header**

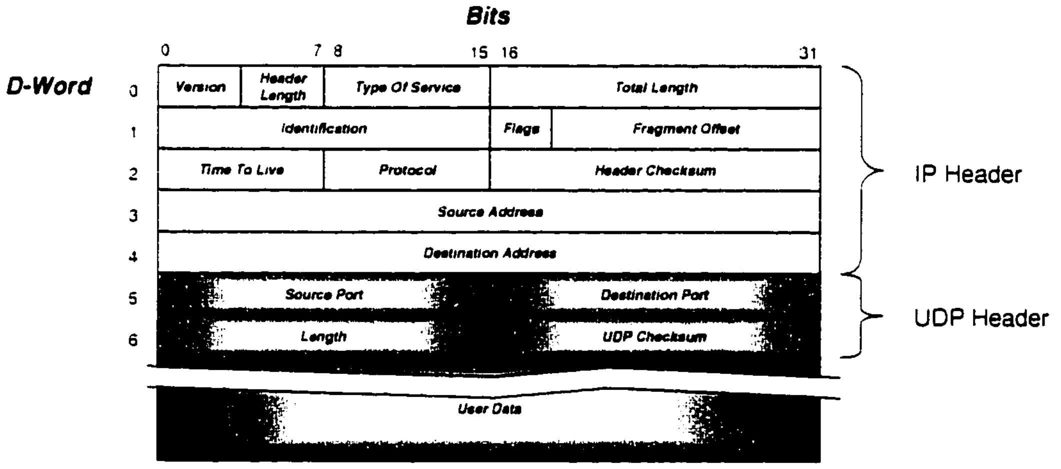


Figure 2.2 **UDP packet format with IPv4 header**

intended for reliable communication, there is more information than is required to be conveyed. Every TCP/IP packet has a 32 bit sequence number, and a 32 bit acknowledgment number and together the two constitute the handshake for reliable packet delivery. A problem arises because in most Internet transfer models the connection is asymmetric (i.e. like web surfing), with the bulk of the information flowing in one direction. The result is that a 40 byte TCP/IP acknowledgment packet containing no user data must be sent in the opposite direction on a 1:1 basis for every information bearing TCP/IP packet. As will be seen, this is significant.

The following figure presents a summary of Internet packet sizes as collected by the

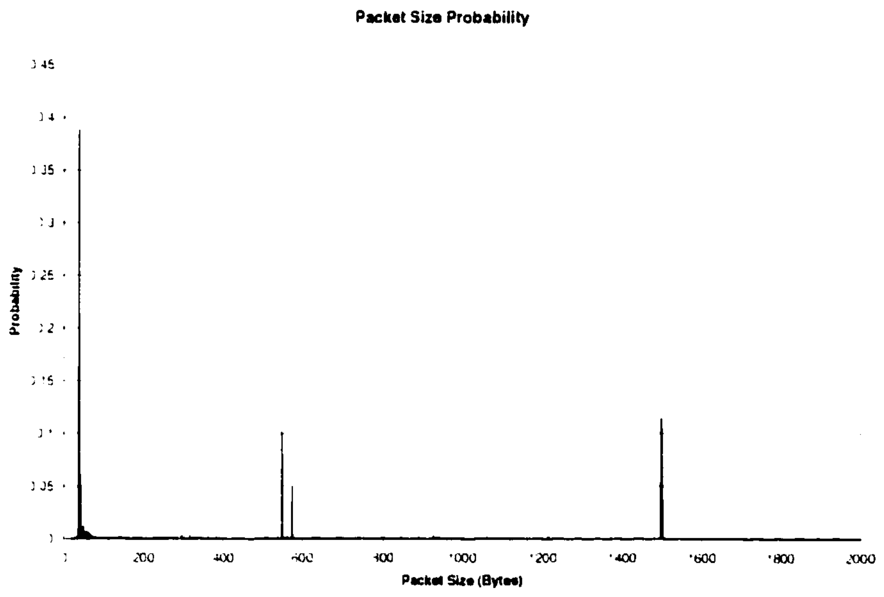


Figure 2.3 **Internet packet size probability distribution**

National Laboratory for Applied Network Research over a one minute interval on an OC-3 (155 Mb/s) link on MCI's backbone[19]. As can be seen, there are five major packet size modes. These are listed in Table 2.1.

Major Packet Mode (bytes)	***Probability***
40	**0.39**
44	**0.06**
552	**0.11**

Table 2.1 **Probability of occurrence of major packet modes**

Major Packet Mode (bytes)	Probability
576	0.05
1500	0.12

Table 2.1 **Probability of occurrence of major packet modes**

If these major modes are deleted from the histogram, the remaining packet size distribution can be observed in Figure 2.4. As can be seen, while there is a variety

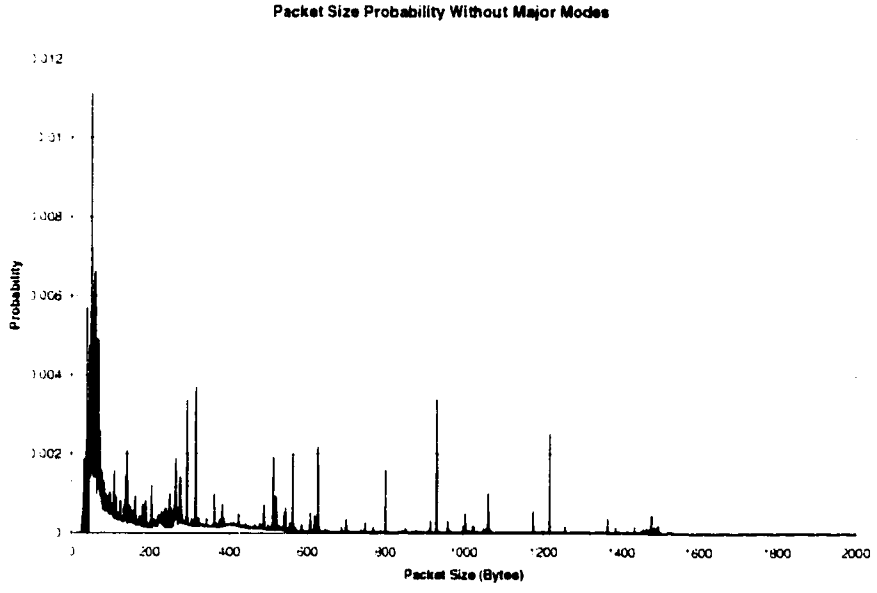


Figure 2.4 **Internet packet size distribution after major mode removal**

of minor modes, the mass is pretty much distributed between 20 and 600 bytes in length. Interestingly, the largest packet seen is only 4352 bytes in length, despite the fact that the maximum allowable packet size is 65535 bytes[†]. Using the complete data set, the average packet size is 354 bytes, with a complete 39% of these packets

[†] The major mode at 1500 bytes is due to this being the upper bound on Ethernet packet sizes.

being 40 byte TCP/IP ACK packets. This implies that a minimum of 78% of all packets on the Internet are TCP/IP based[†].

2.3 The ATM "Cell Tax"

In order to transport data traffic, the public telephone companies (telcos) worked to develop a standard that could transport both voice, video, and data traffic simultaneously. This work was driven from the growth of private companies leasing lines to interconnect their networks. Originally this was done using special boxes called CSU/DSUs (customer service unit / data service unit), but since transporting data over a fixed rate leased line is inefficient[‡] there was an incentive for the telcos to build true data networks, and sell data connectivity directly to the businesses. Starting with X.25, this led to frame relay, and eventually ATM (asynchronous transfer mode), which was billed as the "final solution" for integrated networking. ATM promised to offer the ability to transport any type of service simultaneously by employing quality of service guarantees. Unfortunately, while this is now finally true, these promises were made 8 years ago, and it has taken this long for the standards and technology to make this a reality.

[†] Dropped packets, larger frame sizes (the number of packets per ACK) will cause this number to grow

[‡] This arises from the fact that while the peak bit rate in data traffic is usually as large as the channel will permit, and average bit rate is usually quite small. This allows for the possibility of statistical multiplexing, where many data channels share the same peak bandwidth, and collisions are soaked up in buffers.

Figure 2.5 shows the structure of an ATM cell. It consists of a 5 byte header and a

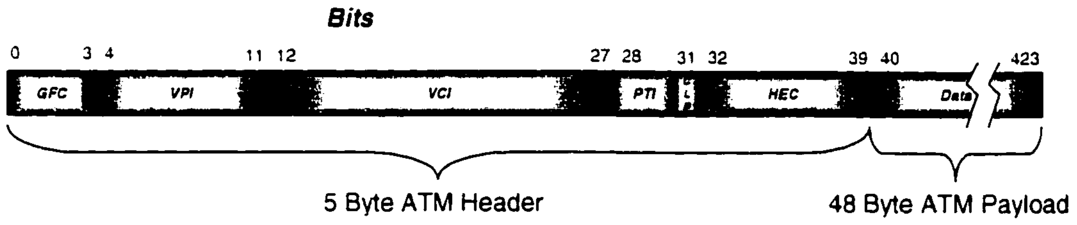


Figure 2.5 **ATM cell structure**

48 byte payload. What is special about it is that the header contains an 8-bit CRC (or HEC - header error check) which provides both the ability to provide a guarantee of the correctness of the bits in the header, and the ability to provide physical layer framing (see Section 3.3). The basic concept is that the ATM cell is like a 48 byte container with dynamic, circuit based routing information contained in the header. Anything to be transported by ATM is cut up into 48 byte chunks and placed into ATM cells, with the start of packet signalled in the header PTI (payload type indicator) field. The process associated with this is referred to as segmentation and reassembly, or SARing. Like all aspects of ATM, there are standards governing how various forms of data can be mapped into ATM using an ATM Adaptation Layer (AAL)[12]. For non-delay sensitive data such as IP packets, AAL-5 is used. Figure

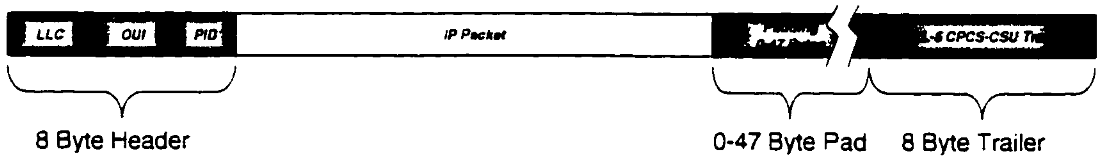


Figure 2.6 **RFC1483 AAL-5 Classical IP Mapping**

2.6 shows the structure of the classical mapping for IP packets into ATM cells, using RFC1483[13]. In this mapping the header is used to identify the payload, and the trailer is used for integrity checking (i.e. it contains a CRC-32 and the length of the payload). Since this entire message is cut up into 48 byte chunks to fit into the payload of ATM cells, a pad field is needed to take up the slack for messages that are not an even multiple of 48 bytes. Unfortunately, as shown in Table 2.1, this happens to be extremely inefficient for 40 and 44 byte datagrams which require two ATM cells to transfer the message. The inefficiencies inherent in classical IP mapping for the major packet modes are given below in Table 2.2. The inefficiency

Major Packet Mode (bytes)	*Probability*	*Classical IP mapping inefficiency*
40	0.39	0.58
44	0.06	0.54
552	0.11	0.04
576	0.05	0.08
1500	0.12	0.02

Table 2.2 **Classical IP transport inefficiency (percent ATM payload not used for IP packet)**

(or "tax") was calculated by determining the fraction of the ATM payload actually carrying the IP packet. When this calculation is extended over all of the packets in the distribution in Table 2.3, the average inefficiency imposed by classical IP mapping (ignoring ATM header overhead) is found to be 34% (i.e. 34% of all AAL-5 mapped payload is used for purposes other than transporting the IP packet). If we now consider the SONET/SDH (4 bytes in 90), ATM (5 bytes in 53), and

AAL-5 overhead, the total tax incurred by transporting IP packets using the classical IP mapping in ATM is 43%. Clearly this is an issue[18].

2.4 Packet over SONET/SDH (POS)

In order to combat this inefficiency, and to avoid having to utilize ATM in a packet based network, the POS standard was proposed[15][16]. This standard dispenses with ATM cells and the AAL-5 layer, and instead uses a PPP (point-to-point protocol) wrapper and directly maps the packets into the SONET/SDH payload[†]. This format is shown in Figure 2.7. The PPP (the same PPP used in dial-up

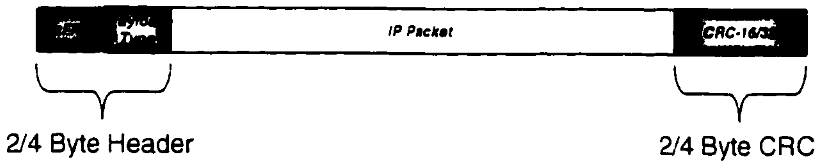


Figure 2.7 **RFC1662 PPP encapsulation**

connections) is used to identify the payload type that is being transported. When the standard was written, there was a 2 byte address and control field reserved in the header. Unfortunately, no one ever figured out what to do with these fields (which default to 0xFF03). As a result, these bytes are often not transmitted[‡]. For payload integrity, a CRC-16 or CRC-32 is applied. This CRC field size is negotiated on connection set-up, or is provisioned.

[†] When people refer to packet over fiber, it is POS that they are usually referring to, which is packet over SONET over fiber.

[‡] This is referred to as a "compressed header".

In order to delineate the packet boundaries, byte escaping is used (versus bit-escaping used on DS-3 (~45 Mb/s) and lower rate connections[†]). The basic concept is that the spaces between packets shall be filled with one or more 0x7E bytes (the HDLC flag). To make this work, all of the natural occurrences of 0x7E within the packet must be changed to something else. In this case, they are replaced with the two byte 0x7D5E combination, with the 0x7D byte indicating that the next byte has been "escaped". All of the naturally occurring 0x7D bytes within the packet must also be escaped. In this case they are replaced with the two byte combination 0x7D5D.

The implication is that the packet will grow in size by an amount proportional to the percentage of 0x7E and 0x7D characters that occur within the packet. If we assume a uniform distribution over all possible bytes, this will cause an average transport tax of roughly 1% (2 bytes in 258). Adding the PPP header we arrive at the final PPP transport taxes of: 6%, 8%, and 10% for 4, 6, and 8 byte PPP overheads. Taking the SONET/SDH overhead inefficiency into account we achieve transport taxes of 10%, 12%, and 14% respectively.

This is far superior to the efficiency of the classical IP mapping into ATM over SONET, and is the reason that so much publicity is surrounding POS (packet over

[†] The reasoning for this is that bit-escaping uses the rule "insert an extra zero after 5 ones in a row". This introduces run length dependencies into the escaping process making it more difficult to pipeline and run in parallel at high speeds, versus byte escaping which only depends on the current byte.

SONET) today. Unfortunately, there are several deficiencies associated with this standard, which became the basis for the work of this thesis. They are:

1) A single bit error in an HDLC flag can cause the loss of an entire packet. This defeats any ability to add error correction at the packet level.

2) The POS (packet over SONET) standard specifies the use of a self-synchronizing scrambler of the form $x^{43} + 1$, similar to the one used in mapping ATM cells into SONET/SDH[9]. Unlike the standard for ATM which specifies scrambling over the payload of the ATM cell, effectively breaking up the stream for 5 bytes in 48, the POS mapping can have a continuous run of up to 65535 bytes of user data. Since self-synchronizing scramblers are susceptible to a malicious attack on their transition density (see Section 4.3), this leaves the door open for hackers to attack the timing recovery and DC balance of the SONET/SDH equipment receiving the packets.

3) The byte escaping process can, in the worst case, double the transmission size of a packet if it is composed solely of 0x7D or 0x7E bytes. This again leaves the door open for hackers to send 64K byte packets consisting solely of these characters, which will double in size upon transmission. This can wreak havoc with switches that employ traffic shapers in their output stage, as they are typically not designed to handle such a "slow-down" for such a long period.

2.5 Multiprotocol Label Switching (MPLS)

The final point to consider in discussing packet transmission is MPLS[20][21]. MPLS is a technique to simplify and reduce the cost of performing the IP lookups and prioritization throughout the network. It involves attaching a tag or label to the packet with information similar to that contained in an ATM header to enable "circuit" based switching. With IP packets that can be up to 64K bytes in length, and with the addition of a tag, greater than 16 bits are needed to describe the total packet length[†]. As a result, any protocol developed must be able to handle this length extension.

[†] Another factor to consider is that a tag may increase the size of the base packet, and shift the mode distribution upward.

Format Synchronization 3-

3.1 Introduction

The purpose of this section is to provide an overview of all of the applicable format synchronization techniques, and discuss the merits of each method.

3.2 Digital Frame Boundary Delineation on Point-to-Point Links

Format synchronization is the process of identifying a unique position in a stream of information. Format synchronization delineation in point-to-point digital transmission systems is used to find frame or packet boundaries in a bit or byte stream. Typically this function occurs immediately after the physical layer symbol recovery has been performed. For instance, in a SONET/SDH system this function would occur immediately after bit synchronization has been achieved using analog techniques.

The point of format synchronization is that it allows the concept of packets and frames, which in turn allows the ability to attach overhead information such as routing information, link quality information, etc. to either a packet or the link itself. In a byte aligned transmission scheme it also provides byte synchronization.

There are two types of formats used for synchronization: synchronous formats, and asynchronous formats. A figure illustrating the different formats is shown in Figure 3.1.

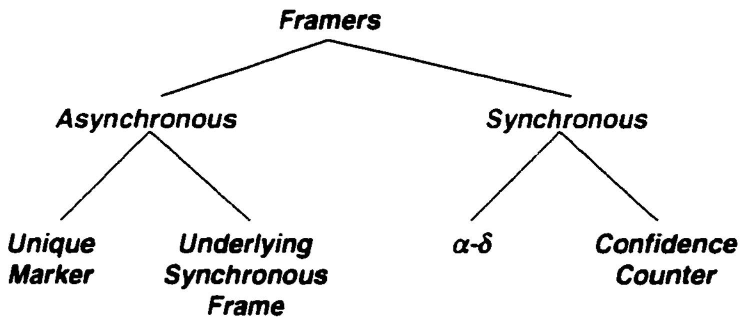


Figure 3.1 **Taxonomy of framers**

The basic difference is that synchronous formats rely on the occurrence of a periodic, known sequence embedded in the transport stream, whereas in asynchronous formats there is typically no known periodicity. This fact inherently makes synchronous formats more robust as each of the individual synchronous frame markers is correlated to all of the other markers. As a result, asynchronous formats are only employed in scenarios where the underlying data being transported exhibits the same qualities (i.e. packet transport).

3.3 Synchronous Formats

As just discussed, synchronous formats employ embedded patterns that occur in a known sequence at known intervals and serve to mark the beginning or end of a frame of data. An example of this is SONET, which employs a 16 bit sequence (the A_1A_2 bytes) that repeat at 125 μs intervals, and serves to mark the start of a SONET frame. A SONET frame for an STS-1 is shown in Figure 3.2. As can be seen there

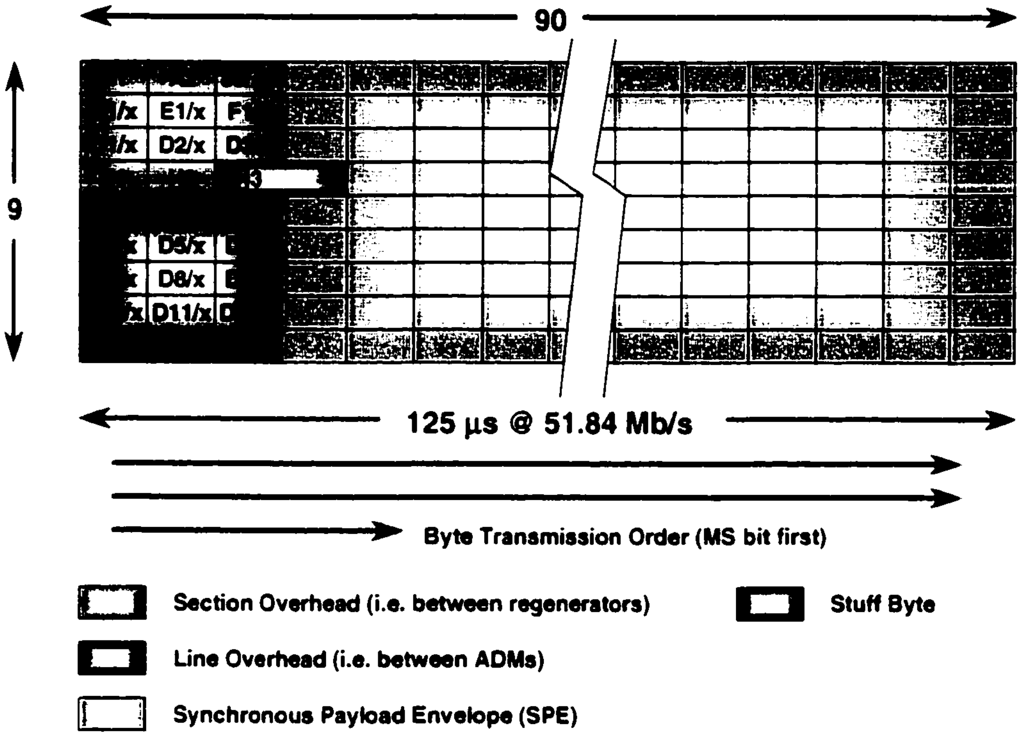

90
9
E1/x
D2/x
D5/x
D8/x
D11/x
125 µs @ 51.84 Mb/s
Byte Transmission Order (MS bit first)
Section Overhead (i.e. between regenerators)
Stuff Byte
Line Overhead (i.e. between ADMs)
Synchronous Payload Envelope (SPE)

Figure 3.2 **SONET STS-1 frame structure**

is a fixed frame length of 125 µs, and the frame format contains overhead bytes as well as payload bytes. The concept here is that information is carried within the payload, and the SONET frame structure provides byte alignment for the payload, as well as overhead information.

In general, synchronous formats using equal length frames are employed in point-to-point transmission systems to allow the periodic insertion of link overhead information such as network maintenance communication channels, error monitoring information such as CRCs and BIPs (bit interleaved parity), etc. The biggest strength that equal length frames have is in their ability to reliably identify the frame boundaries by integrating the knowledge contained in all of the previous frame markers. Although the framing pattern itself is usually not unique from the data, the fact that this pattern repeats at known intervals forever is almost assuredly unique (assuming a correct frame design to defeat mimics in the payload data).

Typically this is done using an α–δ framing algorithm with a state diagram as shown in Figure 3.3. Here the framer starts off searching on a bit-by-bit basis for an

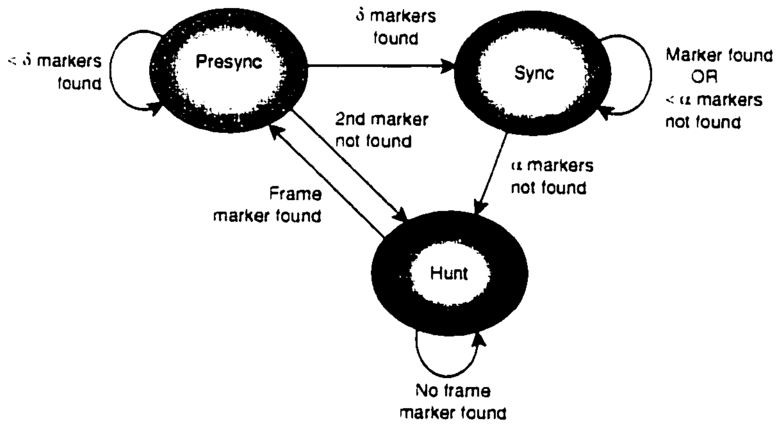


Figure 3.3 **Standard α–δ framer**

occurrence of the framing pattern (the frame marker). As soon as it finds this occurrence, it transitions into the presync state, where it must receive δ consecutive correct frame markers at the correct intervals. Upon receiving the δ^{th} correct frame marker, it transitions to the sync state, where it remains until receiving α incorrect frame markers in a row, upon which it transitions back to the hunt state. This is an ad hoc strategy which has been widely implemented in a variety of applications, and is referred to as "search, check, lock"[23].

If we assume that the frame marker is a fixed length pattern of L_P bits and repeating every L_F bits, and the probability of a bit error is BER, we can make the following observations:

1) Assuming randomly distributed payload data, the probability of a false frame marker is 2^{-L_P}.

2) The probability of correctly receiving a frame marker in the presence of noise is $(1 - \mathrm{BER})^{L_P}$.

Given this, we label the transitions in Figure 3.3 and construct a framer suitable for analysis as shown in Figure 3.4. Here the presync and sync states have been duplicated with their "evil twins": false presync and false sync, and the α/δ states have been explicitly expanded. This transition diagram then allows us to apply Mason's rule to determine characteristics such as mean time to frame, and the probability of false frame, etc.[25].

The typical way that this is done is to express the transitions in terms of both transition probabilities and delay operators measured in the lowest common delay element (such as a bit, or a byte, etc.). For instance, if we are describing an ATM framer operating on a byte synchronized link, $T_{11} = p_{11}D^1$ and $T_{21} = p_{21}D^{53}$,

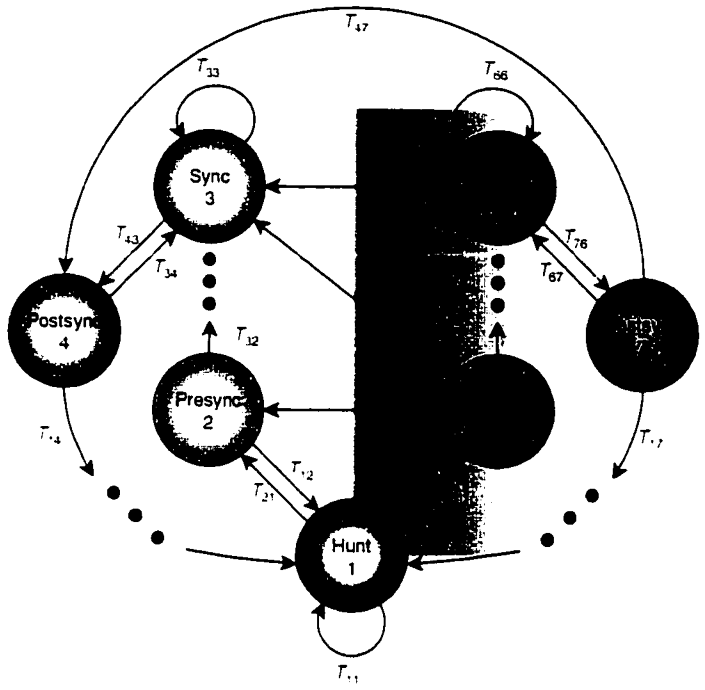


Figure 3.4 **Analysis state transition diagram of α–δ framer**

where D is the lowest common delay element. In this case, the hunt is done on a byte-by-byte basis, whereas the remainder of the transitions are done on a cell-by-cell basis. Once this is done, the average time to achieve frame (and other properties) can be determined explicitly by realizing that the operator:

$$\left.\frac{\partial}{\partial D}\right|_{D=1} \tag{3.1}$$

will translate a transition probability expression into a weighted sum of delays[†].

[†] This can be seen by observing that the output of the equation will be a series of probabilities and exponentiated delay terms. Applying this operator creates a weighted sum of delays as:

$$\left.\frac{\partial}{\partial D}pD^{N}\right|_{D=1} = Np$$

It should be noted that the delays associated with these operations can be written directly into the edges of the transition diagram if the packet sizes or duration of the operation is strictly known. Otherwise, the transition probabilities can be written so as to reflect the uncertainty of an operation, and all of the delays become the delay associated with each cycle, where the cycle time is typically a value of one.

3.3.1 Confidence Counters

α–δ framers are to some extent a subset of the more general technique of confidence counters. Like the α–δ framer, a confidence counter is a saturating counter that increments for every correct occurrence of a frame boundary marker, and decrements for every missed occurrence of a frame boundary marker. The general concept (again like α–δ framers) is that once the confidence counter has reached a certain value, the in-frame condition is declared. At a certain value above this point, the counter saturates. Similarly, there is a point below the in-frame point at which loss of frame is declared (i.e. there may be hysteresis), and the counter is reset. An example of this sort of counter is shown in Figure 3.5. This technique is based on the sequential probability ratio test developed by Wald in 1947[24]. It functions as follows:

1) The confidence counter starts at zero, and for every correct instance the frame boundary marker found, it is incremented by one.

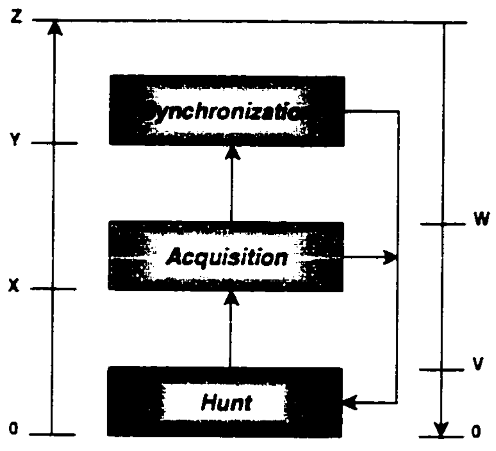


Figure 3.5 **General confidence counter**

2) Once it has reached the verification state, any missed occurrence of the frame boundary marker causes the counter to be reset to zero.

3) Once the confidence counter reaches the value of "X", the verification state is entered. The state machine stays in this state until the counter either reaches the value of "Y", in which case synchronization is declared. If the counter falls below the value of "V", the counter is reset, and the state machine transitions back to the hunt state.

4) Once the confidence counter reaches the value of "Y", synchronization is declared and the counter will continue to increment until it saturates at the value of "Z". The state machine stays in this state until the value of the confidence counter drops below the value of "W", in which case the counter is reset, and the state machine transitions back to the hunt state.

As can be seen, this is very similar to the operation of an α–δ framer, with some additional states. An example of this type of state machine is used in the $x^{31} + x^{28} + 1$ ATM distributed sample scrambler acquisition engine[9]. Typically this is a more flexible state machine, and offers more ability for fine tuning under specific error conditions. For more details the reader is referred to [23].

3.3.2 Error Correction

An additional function that synchronous frames provide is the ability to handle block error correcting codes such as Reed-Solomon coding. These codes are based on a known, usually fixed, block size. The block size is usually conveniently chosen to align with the synchronous frame format, with the check bits being part of the frame overhead[†].

3.3.3 Other Modifications

Another popular modification to synchronous frame formats is the use of super frames. This concept involves using a wrap-around counter in the frame overhead to distinguish between frames. This allows different overhead to be carried in different "sub-frames", and reduces the total overhead associated with the link. A common example of this is the SONET VT (virtual tributary) framing mechanism which uses a four frame super-frame[7].

[†] For more information on block codes, etc. see Appendix C.

Another variant of this is the use of a distributed frame boundary marker. A good example of this is T1 (1.544 Mb/s) framing which uses a 12 frame superframe with one bit per frame, and a 100011011100 frame word spread over these 12 frames[26]. When this was developed, bits were at a premium and this technique afforded the minimum amount of overhead.

3.3.4 Parallel Framers

A final enhancement that can be made to framing algorithms is the use of parallel framers. The mode of operation is to have more than one framer looking for frame sync simultaneously, so that if one framer is off chasing a ghost frame marker, the other framers will still be able to detect the real frame marker. The only rule associated with this enhancement is that no more than one framer can chase the same prospective framer marker, and the first one to reach frame sync wins.

While parallel framers increase the complexity of the receiver, they dramatically improve mean-time-to-frame (MTTF) performance. This is especially true when the frame markers aren't particularly unique from the surrounding data, or in cases where the BER is bad and the frame markers are far apart.

3.4 Asynchronous Frames

Asynchronous frames occur in situations such as packet boundary delineation. Perhaps the most popular example of this is bit-sync HDLC which uses the following set of rules:

1) Fill all space between packets with 0x7E

2) Escape all packet data using the rule "Insert one zero after every occurrence of five ones".

While there are many variants of this (for reasons such as interpacket DC balance, etc.) this example illustrates all of the major tenets of asynchronous framing:

1) No fixed underlying frame structure.

2) Localization of information - the identification of a packet boundary is provided by the immediate information, with no a priori indication.

What asynchronous framing provides is a convenient means of passing isolated and variable length packets between points, with no a priori indication that a packet is going to arrive. Since this is a major attribute of packet networks, bit-sync HDLC is very popular. For instance, aside from being used on all of the PDH (T1/E1) overhead signalling channels, it is the packet boundary delineation method that dial-up modems use in PPP connections[†].

All of this flexibility though comes at a price. By localizing the information indicating the start and end of packets, this type of asynchronous framing is not very robust with respect to bit errors. For instance, a single bit error occurring during interpacket fill which will result in an erroneous false packet (i.e. change the

[†] Currently this is the most common dial-up protocol used.

interpacket fill byte to anything else, and it appears as a single byte packet). As well, a single bit error between back-to-back packets separated by only one 0x7E will result in the loss of both of the packet boundaries. This greatly degrades the BER performance, as this type of frame boundary delineation tends to only be used in low BER channels, or in channels that have been error corrected.

The other common example of asynchronous framing is byte-sync HDLC, which is used for packet over SONET (POS)[17]. Here the rules are slightly more complicated:

1) Interpacket fill is still 0x7E

2) Every 0x7E in a packet is replaced by 0x7D5E[†]

3) Every 0x7D in a packet is replaced by 0x7D5D

Like it's older brother bit-sync HDLC, this framing mechanism suffers from the same susceptibility to errors. The other problem that both of these suffer from is the fact that the packet will increase in size by some amount that is a function of the user data. This can be a problem, as a malicious user can send a long packet that will essentially double in size (for byte-sync HDLC). This can cause problems for traffic shaping circuits which to some extent rely on a constant bit rate on the line.

[†] The concept is that the "escaped" byte is flagged with 0x7E and XORed with 0x20. Originally this was designed to handle ASCII control characters which have a value of less than 0x20.

Scrambling Overview 4-

4.1 Introduction

The purpose of this section is to provide an overview of the various scrambling techniques, their relative merits, and the synchronization methods applicable to them.

4.2 Scramblers

Scramblers are implemented in digital transmission systems to attempt to randomize and DC balance the final data stream that is transmitted. This is done primarily for the following reasons:

1) Receivers typically have to recover the symbol clock, and to do this require transitions in the data stream.

2) Many receivers employ AC coupling in their front-ends to simplify the circuit design. An example of an AC coupled optical receiver is shown in Figure 4.1.

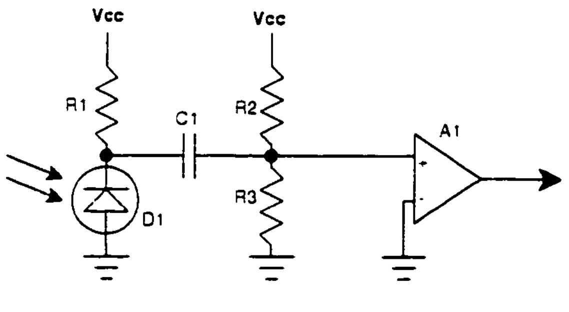


Figure 4.1 **AC coupled optical receiver**

As can be seen in this circuit, resistors R2 and R3 serve to translate the received signal to the midpoint of the comparator's input. Without a DC balanced signal, this input will not be optimally centered at the comparator's input, resulting in sub-optimal performance of the receiver.

A similar effect occurs in copper based transmission schemes which typically employ transformers in their transmitters and receivers for impedance matching, ground isolation, and protection purposes.

3) In binary optical transmission, a zero is characterized as "no light". Unfortunately a loss of signal failure is also characterized as "no light". Given a long enough string of zeroes in such a system (such as SONET), it is possible to falsely declare a loss of signal and in turn implement a protection switch[†]

Because of these issues, scrambling is implemented to attempt to randomize the data such that it's statistics satisfy the requirements of many transitions, DC balance, and no long strings of zeroes. As well, the transmission scheme is typically engineered in such a way as to ensure that the susceptibility of the scrambling and frame scheme to malicious attack is minimized. For instance, in SONET/SDH, all of the various channels are TDM (time division multiplex) interleaved so that the time sequence of data appearing on the line is a composite of many different users.

[†] Most SONET/SDH systems employ line protection on their optical interfaces, which takes the form of a redundant optical interface connected to another optical line, which is diversely routed (to protect against fiber cuts).

There are two major classes of scramblers in use today: self-synchronizing and independent. In the independent category there are three sub-types: set/reset scramblers, distributed sample scramblers, and frame synchronized scramblers. Their taxonomy is shown in Figure 4.2 and their various attributes are discussed below.

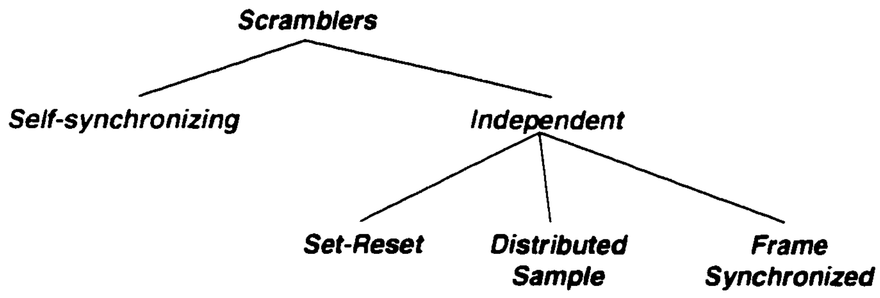


Figure 4.2 **Taxonomy of scramblers**

4.3 Self-Synchronizing Scramblers

Figure 4.3 illustrates the format of a generic self-synchronizing scrambler, and

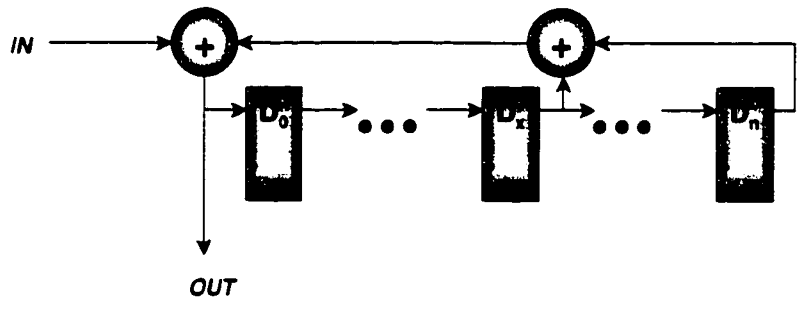


Figure 4.3 **Generic self-synchronizing scrambler**

Figure 4.4 illustrates a generic self-synchronizing unscrambler[27]. As can be seen they are very similar to each other, and in fact their operation can be viewed as a

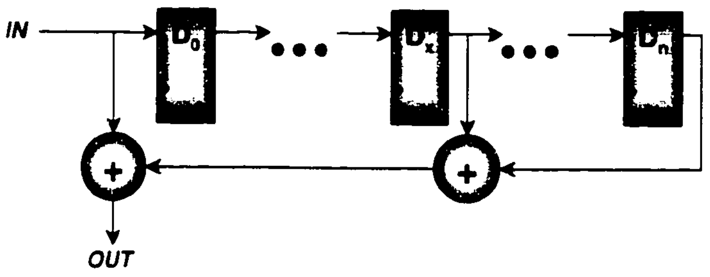


Figure 4.4 **Generic self-synchronizing unscrambler**

finite field version of pole-zero cancellation, where the scrambler is the all-pole filter, and the unscrambler the identical zeroes.

The beauty of self-synchronizing scramblers is the fact that they self-synchronize: no transmission of state information from the transmitter to the receiver is required. As well, they are physically simple to implement, and synchronize after n bits of data (where n is the length of the scrambler) have passed through them.

4.3.1 Self-Synchronizing Scrambler Issues

There are two major problems associated with self-synchronizing scramblers. The first is error multiplication. Every time a bit error occurs in the receiver, this same bit returns in error at every tap (one per polynomial term) in the unscrambler. As a result, scrambling polynomials of the form $x^n + 1$ are used, as they produce two errors for every one error: an effective doubling of the bit error rate. While this seems bad, it is the best that can be done with this form of scrambler.

The second problem with self-synchronizing scramblers is their ability to be maliciously attacked. What this means is that a malicious user will attempt to input a string of data such that the final output data stream contains no transitions, or is heavily non-DC balanced (a condition similar to no transitions). The susceptibility of self-synchronizing scramblers arises because the state of the scrambler is a function of the user data, and while the user does not explicitly know the state of the scrambler at any given point, given knowledge of the system design, the user will know the scrambler's behavior.

Consider the scrambler shown in Figure 4.3. The scrambled output in the presence of a string of zeroes on the input will just be the scrambler repeated over and over. If the scrambler polynomial is primitive, the length of the output sequence will be maximally long[7][28][29]. On the other hand, the minimal length sequence will be generated by polynomials of the form $x^n + 1$. These will have a sequence of length n (i.e. the LFSR implementation of them becomes a barrel shifter)[30]. Unfortunately, these polynomials (especially $x^{43} + 1$) tend to be the most popular, as they are the easiest to implement.

It is polynomials of this form that are also most susceptible to attack. If one inputs a string of zeroes, the output will just be the contents of the shift register repeated over and over. If one's transmission scheme allows a single user to have access to the entire payload of the digital link, the door is open for malicious attack.

[7] A linear feedback shift register built using a primitive polynomial of order n will generate a maximal length sequence of length $2^n - 1$. This is because the linear feedback shift register is the hardware equivalent of long division, and since a primitive polynomial has no factors (i.e. it is equivalent to a prime number), repeated division of any value in the shift register will cycle through all possible values except 0. For more information see Appendix C.

In general, it can be shown that the output power spectral density is a function of the input sequence, and that skewing the zeroes or ones density can directly alter the spectrum in the vicinity of the clock (the half symbol rate - $f_s/2$)[30]. Specifically for binary waveforms with symbols $c_n \in \{0, 1\}$:

$$S(f) = \frac{|\mathrm{sinc}(f)|^2}{4T}\left(\frac{1-(1-2\beta)^{2L}}{1+(1-2\beta)^{2L}-2(1-2\beta)^{L}\cos(2\pi f PT)}+\frac{1}{2}\right) \tag{4.1}$$

where $S(f)$ is the power spectral density, β is the logic one probability, P is the period of the scrambler's impulse response, L is the number of ones in the period, and T is the bit interval. For $\beta \neq 0.5$, it can be seen that the power spectral density has peaks and valleys that in the limit of all zeroes or all ones become line spectra.

This can be seen intuitively from the discussion on how the state behaves in the presence of a string of zeroes and a string of ones.

4.3.2 Method of Attack

The method of attack is as follows:

1) Send a series of packets whose formats consist of n random bits followed by a long string of zeroes, where the length of the string of zeroes is appropriate to cause clock recovery degrade.

2) There are two types of outcomes that can cause failure in clock and data recovery circuits:

a) A large skew in DC balance

b) A low number of transitions

Given a scrambler of the form $x^n + 1$ and the fact that hitting the scrambler with a random vector of length n will initialize the scrambler state to a random vector[†], the probability of having outcome "a" is as follows:

$$p(n, m) = \frac{\sum_{i=0}^{m} D(n, i)}{2^n} = \frac{\sum_{i=0}^{m} \binom{n}{i}}{2^{n-1}} \tag{4.2}$$

where $p(n, m)$ is the probability of a random binary vector of length n having m or fewer ones or zeroes (see Appendix B, Section B.1 for a derivation).

Similarly the probability of outcome "b" is as follows:

$$p(n, t) = \frac{\sum_{i=1}^{m} T(n, i)}{2^n} = \frac{\sum_{i=1}^{t}\left(\sum_{k=1}^{n-2i+1} k^{(2i-2)} + \sum_{k=1}^{n-2i} k^{(2i-1)}\right)}{2^{n-1}} \qquad t \geq 1 \tag{4.3}$$

where $p(n, t)$ is the probability of a random binary vector of length n having t or fewer transitions when repeated ad infinitum (see Section B.2 for a derivation).

[†] After an input sequence of n random bits to a scrambler of the form $x^n + 1$ the scrambler state, the scrambler state will equal the original scrambler state bitwise XORed with the random vector. Since the original scrambler state can be viewed as a random vector (as the prior data into it is unknown), the final state will also be random.

Given eqs. (4.2) and eqs. (4.3), it is possible to calculate what the average number of packets that must be sent before achieving an arbitrary low number of transitions, or an arbitrarily bad DC balance. If the number of packets that a malicious user must send on average to achieve one of these conditions is much less than the average time for loss of frame, etc. due to noise events there is a problem[†].

4.4 Independent Scramblers

An example of an independent scrambler is shown in Figure 4.5. The key difference

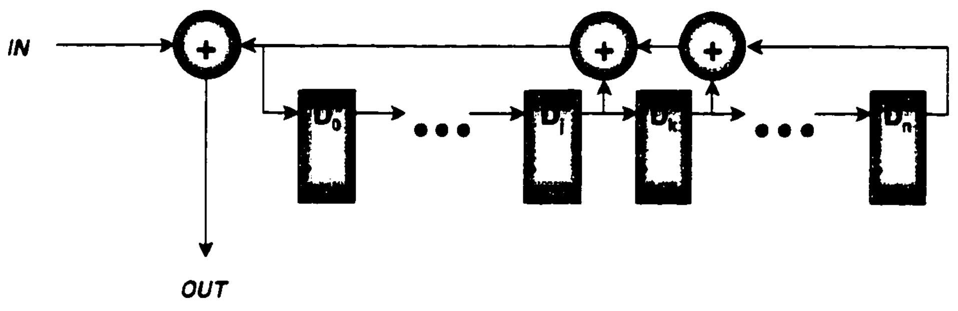


Figure 4.5 **An independent scrambler**

between it and a self-synchronizing scrambler is the fact that the feedback path into D_0 does not have any dependency on the user data it is scrambling. As such, there is no way for the user to influence it. To use a scrambler of this sort, the same circuit is deployed in both the transmitter and receiver. When they are synchronized, one cancels out the other as $a \oplus a = 0$. The issue is that the two ends must be synchronized. The method of this synchronization is what differentiates the sub-types of independent scramblers.

[†] Typical network service availability is on the order of "four nines", or 99.9999% up time.

In terms of polynomial selection, a primitive polynomial is typically used as an LFSR constructed using a primitive polynomial will generate a maximal length sequence with good pseudo-noise properties[31].

4.4.1 Frame Synchronized

A frame synchronized scrambler is the simplest of the independent scramblers and relies on an underlying known frame structure to provide the required synchronization information to correctly align the receiver's scrambler to the transmitter. A simple example of this is SONET/SDH which implements a $x^7 + x^6 + 1$ frame synchronized scrambler. This scrambler is reset on the 1st byte of the 4th column of the SONET frame and then runs through its fixed cycle over and over until it is reset on the next SONET frame.

4.4.2 Distributed Sampling

Distributed sampling scramblers are the second form of independent scramblers. For synchronization, they use a rather ingenious scheme which relies on periodically sampling the output of the transmit scrambler and then sending these samples using an underlying frame format (or other means) to the receiver. The receiver then implements a synchronization scheme which involves comparing its locally generated sample to the received sample. If they don't match, the receiver alters its LFSR (linear feedback shift register) contents by a fixed amount referred to as a "magic number"[32][33].

The best example of a distributed sample scrambler is the one specified in the ITU I.432[9] standard for ATM.

The underpinnings of this approach are as follows:

1) Each LFSR generates a maximal length sequence. By examining the form #2 scrambler (see Appendix C, Section C.2) each bit of the scrambler must output the same sequence with a different offset. The same also holds true for the form #1 scrambler.

2) By the Berlekamp-Massey algorithm[34], this sequence completely describes the scrambler taps and state. Thus, if the sequence is known and the taps are known, the state of the scrambler can be recovered from the output sequence. If there are n flops in the scrambler, n linearly independent samples of the sequence are required.

3) To guarantee linearly independent samples, (i.e. if you view the maximal length sequence stretched out in time, the n samples should never sample the same point in the sequence more than once) make the length of the sequence mutually prime with respect to the sampling interval. In the case of an ATM $x^{31} + x^{28} + 1$ distributed sample scrambler, the length of the sequence is $2^{31} - 1$, which the astute reader may recognize as a Mersenne prime[†], so any sampling interval will

[†] Versus a Fermat prime which takes the form $2^N + 1$.

work. In this particular case, the sampling interval was chosen to be 212 (half of an ATM cell).

4) Given these facts, it is possible to find a fixed number that can be added into the shift register every time the incoming samples fail to match the output samples. After receiving the n^{th} sample, this process will guarantee that the scramblers are in sync[33].

The way to find this number is to realize that LFSRs are linear. Thus the receive scramblers state can be viewed as the sum of the transmit scrambler's state and an error state. Since this procedure works for any error state, we can investigate the case when the error state is such that the first $n - 1$ samples match, and the last sample doesn't. In this case, if we are going to cancel out this error state by adding a fixed value to it, this error state must be equal to the magic number. Given this, it is possible to calculate the magic number for any LFSR.

Consider the case of the $x^{31} + x^{28} + 1$ scrambler. In order to synchronize this scrambler we would have to receive 31 samples. To find the magic number, we just have to find the error state that (on its own in the LFSR) when sampled at the current time would produce a "1", and that for the previous 30 samples produced a "0". Given a transition matrix for the LFSR of $\hat{T}$ and that samples are taken every 212 bits from the most significant bit of the LFSR (bit 30), we can write:

$$\hat{X} = \begin{bmatrix} 0 \ \ldots 0 \ 1 \\ \text{Row30}(\hat{T}^{-212}) \\ \vdots \\ \text{Row30}(\hat{T}^{-(30 \cdot 212)}) \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $\hat{X}$ is the magic number. and Row30($\hat{T}^{-n}$) signifies the entries in the 30th row of the transition matrix $\hat{T}^{-n}$ (which corresponds to the output sample of the LFSR).

A final point to note is that in the ATM scrambler. these samples are conveyed by XORing the two sample that were taken per cell into the upper two bits of the HEC (header error check). Since the HEC is computed over the first 4 bytes of the header. the receiver can also calculate the HEC. and consequently determine the transmitted samples[†].

4.4.3 Set-Reset

The final method of conveying the state of the transmit scrambler to the receiver is to explicitly send it. In order for this to be possible. there must be either a place provided for this in a regular underlying frame structure. or it must be uniquely identified in the transmit stream. The only issue here is that there will inevitably be some sort of delay between when the state was received and when the received scrambler can be updated. However. once one knows the state of an LFSR at one point in time. it is known at any point forward or backwards in time. and this relation is given by the transition matrix of the LFSR. For instance. say one receives the

[†] This technique was used by the author to implement the ATM Tx and Rx blocks for the TDAT042G5. and a patent was applied for on this approach.

transmitter's state $\vec{s}$ now, but require two clock cycles to apply it. In this case, the number one would apply would be $\vec{T}^2\vec{s}$.

Protocol Design 5-

5.1 Introduction

The purpose of this section is to discuss the approach that was used in designing the physical layer framing protocol. The name given to this protocol is SDL (Simple Data Link).

5.2 Goals

5.2.1 POS Deficiencies

The POS (packet over SONET) standard[16] in RFC1619 suffers from the following deficiencies:

1) Poor framing robustness with respect to bit errors: a single bit error in the 0x7E flag can generate false short packets, or cause the loss of a packet boundary.

2) Variable transmit packet size expansion: the byte escaping mechanism can effectively double the transmit packet size, potentially causing problems in traffic shaper circuitry.

3) Error multiplication: the $x^{43} + 1$ self-synchronizing scrambler doubles the received bit error rate, as every error shows up again 43 bits later to corrupt another bit.

4) Susceptibility to DC balance and transition density attacks: self-synchronizing scramblers are susceptible to malicious attacks on high-speed receiver front-ends (see Appendix D).

5) No native link quality estimation ability: the byte-escaped HDLC (high level data link control) scheme provides no ability to estimate link BER.

6) No ability to extend the transport mechanism to run over fiber: the byte-escaped HDLC scheme (as the name implies) relies on an underlying framing structure to provide byte synchronization.

5.2.2 New Protocol Goals

Based on these deficiencies the goals for this new protocol are as follows:

1) Robust packet boundary delineation for packets from 4 bytes to 65535 bytes in length with fast reframe time

2) No packet size expansion

3) No error multiplication

4) Immunity to malicious attacks on DC balance and transition density.

5) Ability to estimate link quality

6) Ability to run over fiber directly (i.e. bit-sync ability)

7) Ability to run at high speed (i.e. easy to implement)

8) Ability to operate reliably in BER environments up to 10^{-3}.

Given these goals the following sections describe the design of the protocol.

5.3 Protocol Design

5.3.1 Robust Framing

In order to make the delineation of packet boundaries robust there are two approaches that can be used:

1) Have a unique marker. The longer the marker the more unique it is, but the more susceptible it is to corruption by bit errors. As well, an escaping or encoding process is required for the packet payload to eliminate any occurrences of the unique marker. This guarantees expansion of the packet size. As a result this approach was abandoned.

2) Have a repeated frame structure and use pointers to identify the packet boundaries. Similar to SONET/SDH[†], this method would provide the ability to robustly identify the packet boundaries. An example of this line of thought is shown in Figure 5.1.

[†] i.e. SONET/SDH uses an 8 KHz frame structure that contains a pointer to a floating payload. However, this payload is roughly fixed in size (the difference is due to clock skew), which makes this method a good choice.

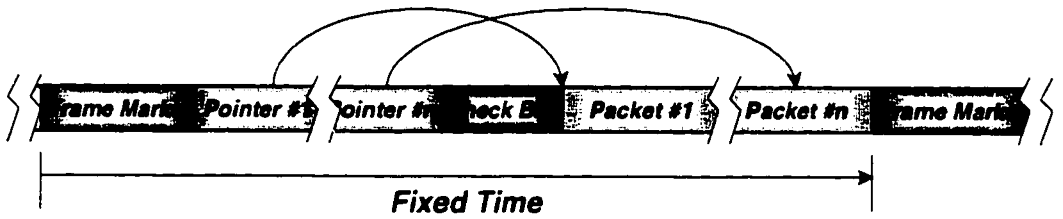


Figure 5.1 **Fixed frame with error corrected pointers**

Here there is a fixed underlying frame structure which provides the ability to use an α–δ framer or a confidence counter. Associated with this would be a series of pointers with error correction applied to them that would point to the beginning of any packet boundaries contained within the local frame payload. Unfortunately, the range of packet sizes that must be handled makes this method untenable.

Given that neither of these two methods would function properly in this environment, a new method was required. In conjunction with Bell-Labs[18], a novel variant of the method shown in Figure 5.1 and ATM HEC framing was developed.

The basic concept is to use an error corrected pointer to delineate the packet boundary, but to also use this error corrected pointer (as in ATM) to identify this collection of bytes as a pointer. For instance, if one has an 8 bit pointer and an 8 bit CRC, there is a unique mapping between the pointer and the CRC[†]. As a result the

[†] Since a CRC is the remainder by division in GF (2) of a generator polynomial, this is only true if the size of the field over which the CRC is calculated is less than the order of the polynomial. For more details on this see Appendix E.

chance of two bytes of random data emulating a valid pointer and CRC is $1/256$. This concept is similar to the α–δ framing algorithm used in ATM. The difference is that in ATM HEC framing the spacing between header occurrences is fixed, whereas in this concept, the distance can vary from 4 to 65535 bytes. An example of this framing concept is shown in Figure 5.2.

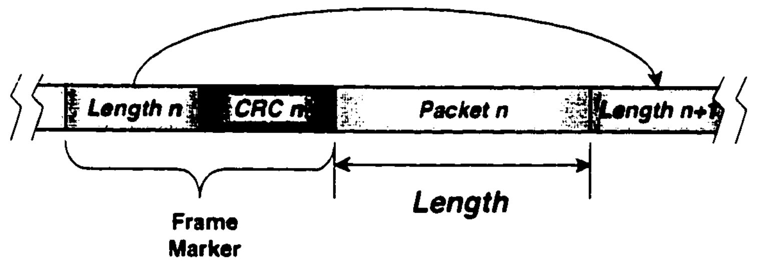


Figure 5.2 **Asynchronous frame marker with pointer**

This algorithm provides the following:

1) Relatively unique packet boundary delineation: The longer the CRC, the lower the chance of false emulation in random data.

2) Error correction capability: The CRC field is capable of performing error correction on the payload field as the remainder can be considered a cyclic code. Given that the CRC is calculated using a primitive polynomial[†], the rough rule of thumb for error correcting capabilities is that for every bit of correction capability, you need one bit to identify the polarity, and $\log_2 N$ bits to identify where in the N bit payload and CRC the errored bit is.

[†] This is often not the case as CRC codes generally are the product of $(x+1)\cdot p(x)$ where $p(x)$ is a primitive polynomial. This is done to guarantee detection of odd length error patterns[36].

3) Distributed framing information: Since each future frame marker is pointed to by the previous frame marker, the ability to use a subset of synchronous framing techniques exists. The reason that the full complement of synchronous techniques is not available is that if a frame marker is corrupted, you are now lost and have to go immediately back into the hunt state. This restriction basically means that only $\alpha-\delta$ framers with α set to 1 are viable (i.e. on the first miss, go back to hunt).

4) Link quality estimation: By implementing error detection and correction in the CRC field, it is possible to estimate the BER of the link by observing the number of errors in the headers over a given interval.

5) Bit sync capability: Since the frame header is relatively unique in all byte and bit shifts, it is possible to run directly over fiber, and not rely on an underlaying synchronous frame to provide byte synchronization.

Given these attributes, it is possible through judicious choices of the length field and the CRC field to satisfy the goals stated in 1, 2, 5, and 6 above[†].

5.3.2 Framer and Frame Marker Engineering

Given that we are constrained to using an $\alpha-\delta$ framer with α set to 1 (as mentioned above), there are 3 items that need to be selected: the value of δ, the size of the

[†] Since the bulk of traffic is IPv4, one may ask "Isn't there some way to reuse the length field in the IPv4 header"? The answer is basically yes, but the object is to create a protocol that is payload independent, as it is desirable to be able to transport packets other than IPv4 (for instance MPLS tagged IPv4).

length field, and the size of the CRC field. Of these three items, the length is pretty much constrained to be a 16 bit value due to the requirement to handle 65535 byte packets ($2^{16}-1$) which is the maximum IP (Internet protocol) datagram size. So the only factors to engineer are the size of the CRC, and the value of δ.

Below are listed the factors that affect the choice of these two values:

1) The minimum number of bits in the CRC field to provide t bits of error correction is governed by the Hamming bound[35]:

$$2^{L_{\mathrm{CRC}}} \geq \sum_{j=0}^{t} \binom{L_{\mathrm{Pointer}} + L_{\mathrm{CRC}}}{j} \tag{5.1}$$

where L_{Pointer} is the number of bits in the length field, and the number of bits in the CRC field is L_{CRC}. Thus, for a 16 bit length field and single bit correction we would need at least a six bit CRC field.

2) Since the framing mechanism is forced to have α set to one, we need to be able to rapidly reframe, and to have a low probability of false frame. This ideally requires δ set to a low value such as one or two. Given that the probability of false frame is directly related to the chance of finding δ false frame markers, we can state that the probability of false frame is roughly:

$$P_{\mathrm{FF}} \equiv \frac{1}{2^{\delta L_{\mathrm{CRC}}}} \tag{5.2}$$

Given this, and a desire[†] to have the probability of false frame lower than 10^{-9}, we are left with the requirement that δL_{CRC} be at least 30 bits[‡].

3) It is desirable from a circuit implementation perspective to have everything occur on byte boundaries. This would dictate that L_{CRC} be a multiple of eight.

4) It is desirable that the overhead due to framing be as small as possible.

Given these four factors, it is clear that a δ value of 2, and a CRC field size of 16 is the correct choice.

5.3.3 Parallel Framers

Given these choices, the next piece of engineering is to consider if parallel framers are required. This directly affects MTTF, and since MTTF is important, and frame markers can be up to 65535 bytes apart, it was decided to investigate their merit.

The analysis version of the state machine for a single framer is shown in Figure 5.3. As can be seen, this is derived from the general α–δ analysis framer shown in Figure 3.4, with α and δ both set to one. For the parallel framer scenario, the situation gets trickier. In Figure 5.4, the analysis state transition diagram for an arbitrary number of parallel framers is shown.

This diagram can be explained as follows:

[†] This is a rough rule of thumb.

[‡] This also drives the requirement to not have error correction "turned on" in the hunt phase, as the probability of locking on to a false marker goes way up.

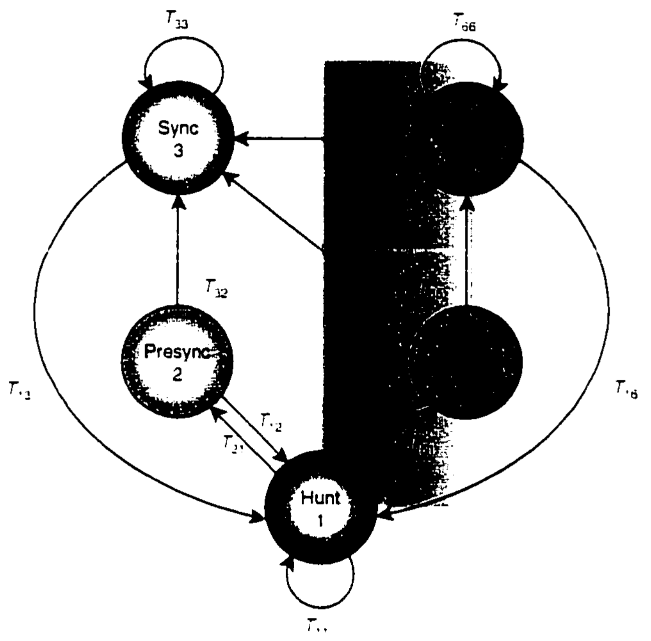


Figure 5.3 **Analysis state transition diagram of single SDL framer**

1) The parallel framers are never allowed to track the same frame marker. This results in only one event per cycle.

2) The discernible states of the N parallel framers can be classified as "one framer chasing something, two framers chasing something, etc. Since there can be only one good frame marker that is being chased, and as many bad ones as there are framers, the state transition diagram is organized into a hierarchy of active framers, with each level being more framers active. The exact composition of the states is labelled using the symbols, G, B, and H, where G indicates a framer chasing a good frame marker, B indicates a framer chasing a bad frame marker, and H indicates a framer in hunt. For instance GB^2H^3 would indicate

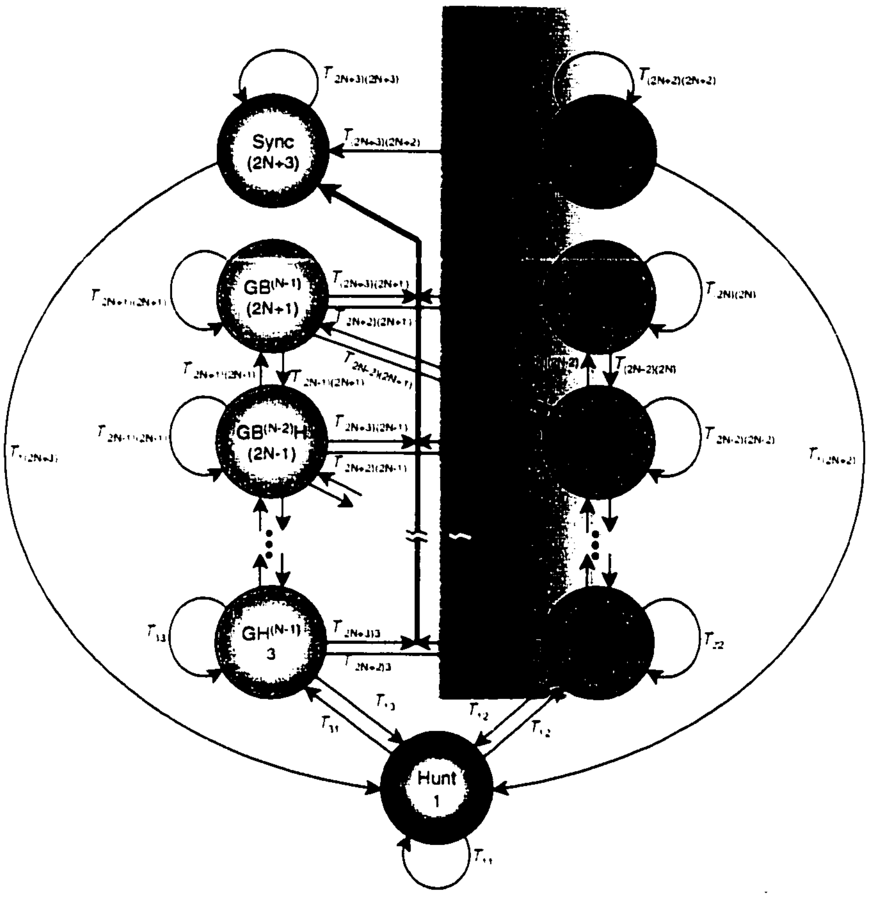


Figure 5.4 **Analysis state transition diagram of N parallel framers**

a 6 framer system where one framer was chasing a good frame marker, two were chasing bad frame markers, and three were idle.

3) The first framer to reach synchronization (after reaching $\delta = 2$) disables all of the other framers. Hence upon loss of synchronization, the state transition is back to "all hunt".

The transition probabilities can be determined as follows[†]:

[†] Note that in all of these probabilities, they are on an "event-by-event" basis (either bits, bytes, etc.), hence the occurrence of L_{AVG} in the probabilities.

Hunt Transitions

1) For a framer in hunt, the probability of finding a false frame marker is[†]:

$$P_{BH} = \frac{1}{2^{L_{CRC}}} = \frac{1}{2^{16}} \tag{5.3}$$

2) For a framer in hunt the probability of finding a good frame marker is:

$$P_{GH} = 2 \cdot \frac{1}{L_{AVG}}(1 - \text{BER})^{L_{\text{Marker}}} = \frac{2(1 - \text{BER})^{32}}{L_{AVG}} \tag{5.4}$$

This is the probability of hitting a packet from a random start position (the factor of 2) and not having the frame marker corrupted.

False Presync and False Sync Transitions

1) For a framer in false presync and false sync, the probability of transitioning to hunt is:

$$P_{HB} = P_{HF} = \frac{1}{2^{L_{\text{Max}}}/2}(1 - P_{BH}) = \frac{1}{2^{31}} - \frac{1}{2^{47}} \tag{5.5}$$

which is the probability of hitting the "end" of the false frame (remember that this is a random length, hence the $\frac{1}{2^{L_{\text{Max}}}/2}$) and having it not emulate a frame marker.

2) For a framer in false presync and false sync, the probability of transitioning to false sync is:

[†] See Appendix E for a discussion of this.

$$P_{FB} = \frac{1}{2^{L_{\text{Max}}}/2}(P_{BH}) = \frac{1}{2^{47}} \tag{5.6}$$

which is the probability of hitting the end of the false frame and having it emulate a frame marker.

3) For a framer in false presync and false sync, the probability of transitioning to sync is:

$$P_{SB} = \frac{1}{2^{L_{\text{Max}}}/2}(P_{GH}) = \frac{1}{2^{31}}\left(\frac{(1-\text{BER})^{32}}{L_{\text{AVG}}}\right) \tag{5.7}$$

which is the probability of hitting the end of the false frame and having it be a good frame marker.

Good Presync and Good Sync Transitions

1) For a framer in good presync and good sync, the probability of transitioning to hunt is:

$$P_{HG} = P_{HS} = \frac{1}{L_{\text{AVG}}}(1-(1-\text{BER})^{L_{\text{Marker}}}) = \frac{1-(1-\text{BER})^{32}}{L_{\text{AVG}}} \tag{5.8}$$

which is the probability of hitting the end of the frame and having a corrupted frame marker.

2) For a framer in good presync, the probability of transitioning to sync is:

$$P_{SG} = \frac{P_{GH}}{2} = \frac{(1-\mathrm{BER})^{32}}{L_{\mathrm{AVG}}} \tag{5.9}$$

In all of the cases above, the delay associated with each edge in the transition diagram is the time taken to process one byte. This is because the transition probabilities are set to deal with events on a byte-by-byte basis and not on an event by event basis.

5.3.3.1 Signal Flow Equations from a Transition Matrix

Typically at this point, the transfer function from the hunt state to the sync state would be constructed using Mason's Rule as described in Section 3.3, which would then allow the determination of the mean time to frame (MTTF), etc. However, the concept of applying Mason's Rule to Figure 5.4 is rather heinous. Consequently, a formulation of the transfer function from the transition matrix is desired. In this way, a formulation in MATLAB can be constructed to calculate the desired framing parameters automatically for an arbitrary number of framers.

5.3.3.2 Examination of Signal Flow Graph Representations

A signal flow graph represents summation by convergence of edges at a node, and multiplication by values assigned to edges[37]. For instance the equation $y = mx + b$ would be represented in a signal flow graph as shown in Figure 5.5.

Working from this we can write the system of equations that describe the signal flow graph in terms of the transition matrix as:

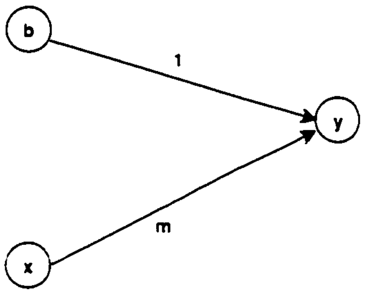


Figure 5.5 **Signal flow example of** $y = mx + b$

$$\hat{T}\hat{n} = \hat{n}$$

or:

$$(\hat{T}-\hat{I})\hat{n} = 0 \tag{5.10}$$

Now if we examine the transfer function of a single input, single output (SISO) system[38] described by this system of equations, we can solve for the transfer function if we do the following:

1) Write the system of equations with a single unitary edge leading into the signal flow graph from an arbitrary first node, which has the effect of making the matrix in eq. (5.10) look like:

$$\begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & T_{11} & \cdots & T_{1n} \\ 0 & T_{21} & \cdots & T_{2n} \\ \vdots & \vdots & \ddots & \ddots \\ 0 & T_{n1} & \cdots & T_{nn} \end{bmatrix}$$

2) Label the nodes in the signal flow graph in increasing order with the desired output node as the last node.

3) Solve eq. (5.11) for n_n (the desired output node)[†].

$$(\hat{T}-\hat{I})\hat{n} = \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.11}$$

Thus:

$$\hat{n} = (\hat{T}-\hat{I})^{-1} \cdot \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.12}$$

Using this technique, we can determine the value of the transfer function for any arbitrary transfer function described by the transition equations of a signal flow graph.

5.3.3.3 Matrix Formula for MTTF

MTTF (as described in Section 3.3) can be found from the transfer function by applying the operation described by eq. (3.1). Putting this in the context of eq. (5.12), we have[‡]:

[†] This is equivalent to writing the output node n_n in terms of n_0, then dividing the resulting equation for n_n by n_0 and setting $n_0 = -1$, which results in the transfer function.

[‡] Remember that we have to set the transition probability of staying in the sync state to zero for this operation (i.e. $T_{nn} = 0$) as we want the average time, and not the integral. Otherwise the weighted sum of delays would continue to add in the probability mass that has already reached the sync state.

$$\begin{aligned}\left.\frac{\partial}{\partial D}\vec{n}\right|_{D=1} &= \left.\frac{\partial}{\partial D}(\vec{T}-\vec{I})^{-1}\cdot\begin{bmatrix}-1\\0\\\vdots\\0\end{bmatrix}\right|_{D=1}\\ &= \left.-(\vec{T}-\vec{I})^{-2}\cdot\frac{\partial}{\partial D}\vec{T}\cdot\begin{bmatrix}-1\\0\\\vdots\\0\end{bmatrix}\right|_{D=1}\end{aligned} \tag{5.13}$$

5.3.3.4 Transition Matrix for N Parallel Framers

Given the signal flow graph in Figure 5.4 and the transition probabilities described by eqs. (5.3) - (5.9), we can write the general transition matrix for N parallel framers as:

$$\begin{bmatrix}\text{Input Edge Transitions}\\\text{Hunt Transitions}\\\text{One False Presync/Hunt Transitions}\\\text{One Good Presync/Hunt Transitions}\\\text{Two False Presync/Hunt Transitions}\\\text{One Good, One False Presync/Hunt Transitions}\\\vdots\\\text{False Sync Transitions}\\\text{Sync Transitions}\end{bmatrix}$$

or, filling in the blanks:

$$\hat{T} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
1 & 1-\Sigma_c & P_{HB} & P_{HG} & 0 & 0 & 0 & 0 & 0 & \cdots & P_{HF} & P_{HS} \\
0 & P_{BH} & 1-\Sigma_c & 0 & P_{HB} & P_{HG} & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & P_{GH} & 0 & 1-\Sigma_c & 0 & P_{HB} & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & P_{BH} & 0 & 1-\Sigma_c & 0 & P_{HB} & P_{HG} & 0 & \cdots & 0 & 0 \\
0 & 0 & P_{GH} & P_{BH} & 0 & 1-\Sigma_c & 0 & P_{HB} & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 0 & P_{BH} & 0 & 1-\Sigma_c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & P_{GH} & P_{BH} & 0 & 1-\Sigma_c & 0 & 0 \\
0 & 0 & P_{FB} & 0 & P_{FB} & P_{FB} & P_{FB} & P_{FB} & P_{FB} & \cdots & 1-\Sigma_c & 0 \\
0 & 0 & P_{SB} & P_{SG} & P_{SB} & P_{SG} & P_{SB} & P_{SG} & P_{SB} & \cdots & P_{SF} & 1-\Sigma_c
\end{bmatrix} \tag{5.14}$$

Here the $1 - \Sigma_c$ indicates that the probability of remaining in the state is what's left after all of the probabilities for leaving the state are summed up (i.e. every other entry into the column). Note that the magenta entry is the one that is deleted for the MTTF calculation.

5.3.3.5 MTTF

Now to find the MTTF from this transition matrix, we include a delay variable in the transitions, and after finding the transfer function take the partial derivative of this function and evaluate it with the delay variable set to one (see eq. (3.1)). Since this is a system of linear equations and partial differentiation is also a linear operation, we can apply the delay variable to the transition matrix and apply the partial derivative operator before solving.

5.3.3.6 MTTF Results

Using the pattern displayed in eq. (5.14), a MATLAB program (see Appendix G) was written to solve for n_n in eq. (5.14) for various bit error rates, packet lengths and number of parallel framers. The results are shown below.

Figure 5.6 plots MTTF versus BER at 356 bytes for a range of framers, and Figure

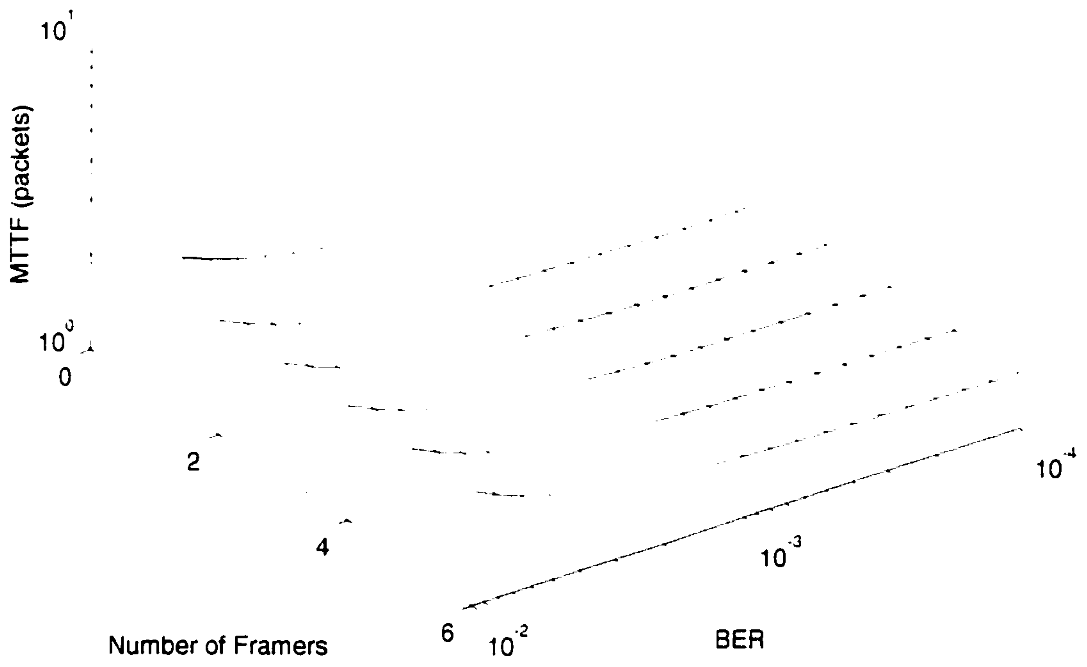


Figure 5.6 **MTTF versus number of framers at 356 byte average packet size versus BER**

5.7 plots the MTTF at a BER of 10^{-3} for 356 byte packets. Figure 5.8 plots MTTF versus BER at 64K bytes for a range of framers, and Figure 5.9 plots the MTTF at a BER of 10^{-3} for 64K byte packets.

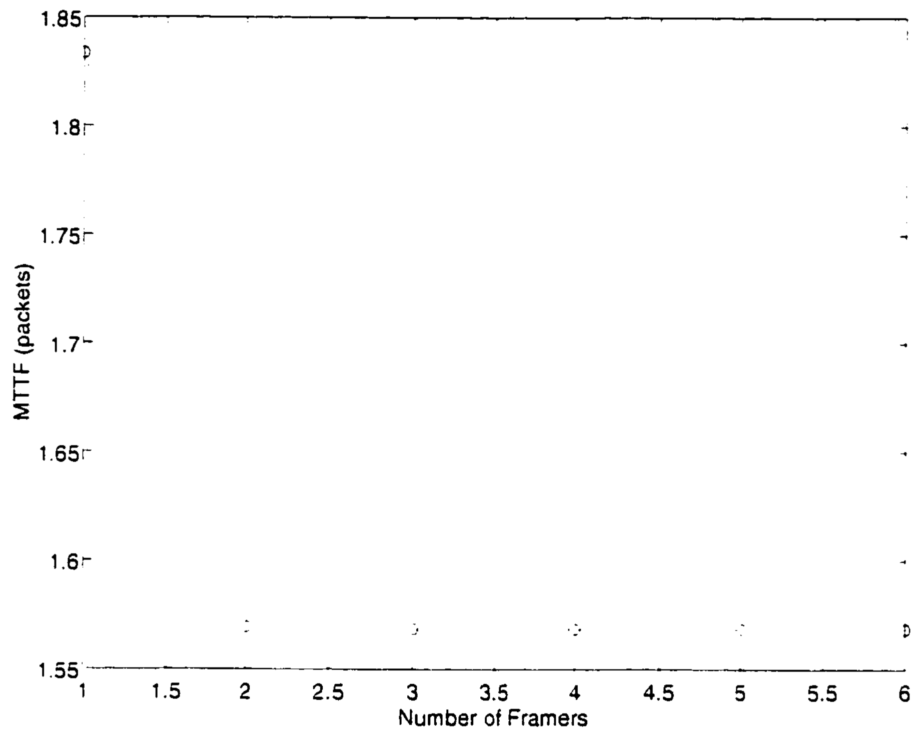


Figure 5.7 **MTTF versus number of framers at 356 byte average packet size at BER = 10^{-3}**

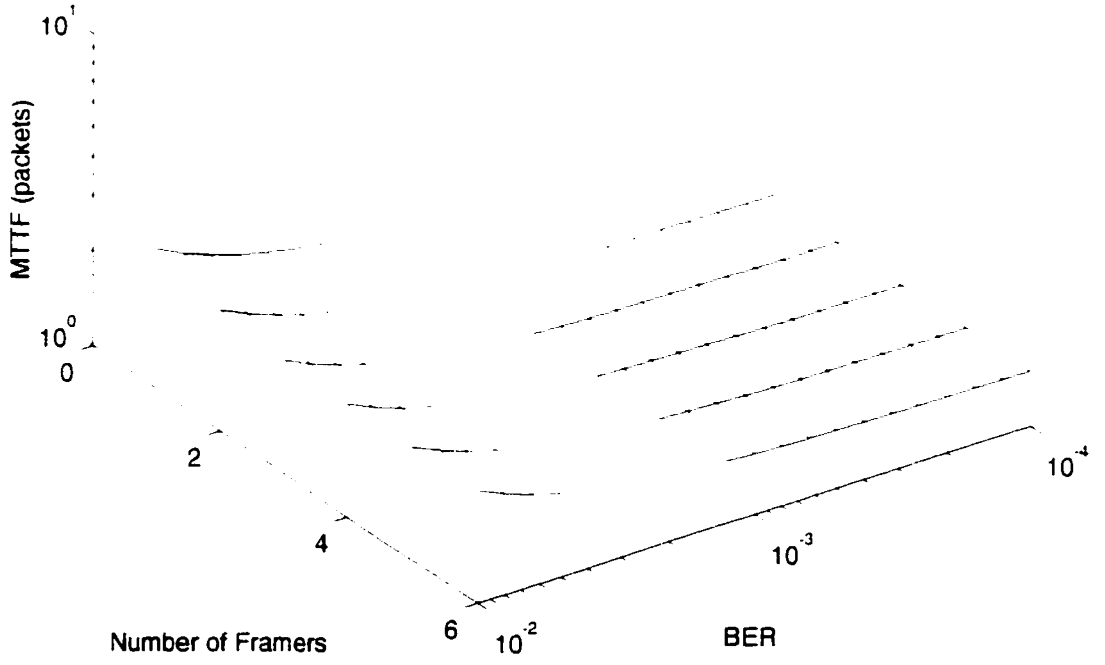


Figure 5.8 **MTTF versus number of framers at 64K byte average packet size versus BER**

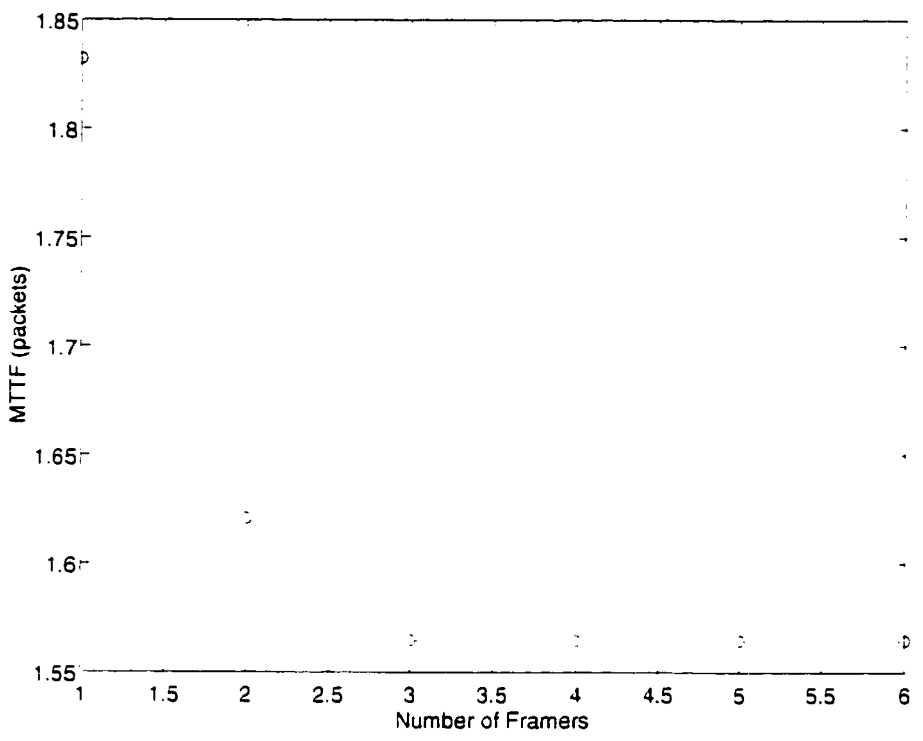


Figure 5.9 **MTTF versus number of framers at 64K byte average packet size at BER = 10^{-3}**

Based on these results, it can be seen that the performance of the protocol is adequate for reasonable bit error rates, and a reasonable number of framers. Based on these results, four parallel framers were chosen. In fact three would have been sufficient, but from an implementation perspective four was just as easy (being a power of two), and in reasonable BER environments even a single framer will give adequate performance[†].

[†] The concern is that while the difference between 1.5 and 2 packets MTTF isn't much, it is the variance that causes this jump that is.

5.3.4 CRC-16

The next choice is the polynomial used for the CRC. Examining the error correction properties of a 16 bit primitive polynomial, there are 15 bits available for bit position location and error indication. Given the relationship in eq. (5.1), and the fact that the header size is 32 bits (16 bit length pointer + 16 bit CRC), we have the capability of correcting up to two header bit errors. However, upon examining the complexity of the syndrome table for double bit correction (528 entries), and the fact that a popular transmission medium would be an optical link (which without forward error correction is shot-noise dominated: i.e. isolated single bit errors), it was concluded that error detection and single bit correction was a better choice. As such, the CCITT CRC-16 polynomial $g(x) = x^{16} + x^{12} + x^{5} + 1$ was chosen. It is a polynomial of the form $(x + 1)p(x)$ (again for enhanced error detection capability) which is important for error detection purposes. Examining its Hamming distance, it was confirmed that this polynomial had single bit correction capabilities.

5.3.5 Null Fill

In order to maintain framing, the SDL protocol must be able to provide framing in the absence of packets. In order to accomplish this, it was decided to insert zero length packets[†] (see Figure 5.10). This results in the insertion of as many of the following headers as is required to bridge the gap until the next packet arrives for

[†] Which is also logically nice, as there is no information available.

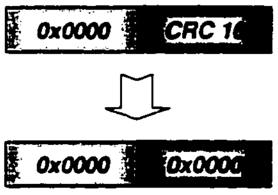


Figure 5.10 **Null fill header**

transmission. The implication of this is that gaps between packets are forced to be multiples of four bytes in length[†].

5.3.6 DC Balance and Transition Density

A more important consideration in the use of the null packet as interpacket fill is the fact that the this packet (which in an idle link is the only packet being sent) is not DC balanced and has no transitions. As is discussed in Section 4.2, this is unacceptable for an AC coupled system. To solve this problem, it was decided to add (modulo 2) a fixed 32 bit number that exhibited both DC balance, and a maximum number of transitions. One obvious choice for this that meets the two aforementioned criteria is an alternating one/zero sequence[‡]. However the issue with this is that it does not allow for the identification of the 32 boundaries of the frame marker. Mathematically what is desired is a length 32, DC balanced

[†] While this may be considered "fortuitous" for high speed parallel implementations, it only becomes useful if the packets themselves are also multiples of 4 bytes in length. An option is to pad out the packets to be multiples of 4 bytes long, however it was deemed that the extra benefit to hardware designers implementing this protocol would be minimal, and all that would be achieved is a decrease in transmission efficiency.

[‡] Another possibility would be to use a DC balanced line code such as 8B/10B coding such as in Gigabit Ethernet (i.e. a byte is mapped into 256 of the DC balanced codes in the 10 bit code space). The issue with this is that it degrades transmission efficiency by 25%.

sequence, with a maximum number of transitions, and minimum autocorrelation sidelobes[†].

To find such a sequence, a computer program was written (see Appendix H) that implemented a search of all possible candidate sequences of length 32. While computing power continues to double roughly every 18 months[40], implementing a brute force search on the autocorrelations of 2^{32} sequences is impractical. However, by applying some simple sieve rules to the problem, the computing time for this problem can be reduced to something quite reasonable[‡]. These rules are:

1) Every sequence has an order inverse and a bit complement which from an autocorrelation perspective are identical. Thus it was decided to only examine new sequences. Computationally, this was implemented as a count from 0 to $2^{32} - 1$ with a check to skip the sequence if the inverse or complement is less than the current count.

2) Don't check non-DC balanced sequences.

3) Keep track of the minimum autocorrelation sidelobe set seen so far, and if a sidelobe in the current sequence exceeds it stop checking.

[†] This is somewhat similar to a Barker sequence[39], which is a sequence that exhibits autocorrelation sidelobes of magnitude one (the minimum). Note that Barker sequences must be of odd length or else the peak is smeared.

[‡] i.e. less than a couple of hours on a Sun Ultrasparc 60

4) Compute the autocorrelation values "backwards": i.e. start at the center of the sequence and work outward, as the larger sidelobes are typically found towards the center of the autocorrelation. This rules out poor sequences more rapidly.

Using this technique the following DC balanced sequences were determined to have the minimum possible autocorrelation sidelobes:

```
00000111100011001101010101101101 = 0x078cd56d
Sum of Lobes = 44 Transitions = 18

01001001010101001100111000011111 = 0x4954ce1f
Sum of Lobes = 44 Transitions = 18

10110110101010110011000111100000 = 0xb6ab31e0
Sum of Lobes = 44 Transitions = 18

11111000011100110010101010010010 = 0xf8732a92
Sum of Lobes = 44 Transitions = 18
```

Given these sequences, which are indistinguishable given the criteria used to evaluate them (and the fact that they are inverted mirrors of each other), the sequence 0xB6AB31E0 was arbitrarily chosen[†]. Its discrete autocorrelation is shown in Figure 5.11.

5.3.7 Scrambling

The next major piece of the protocol that was designed was the scrambling. As discussed in Chapter 4, scrambling is crucial to eliminate the ability of the payload

[†] As an historical note, CRC checks have traditionally dealt with the problem that 0 divided by $g(x)$ is 0 in one of two ways: either preload the LFSR with a value (as is the case of the CCITT CRCs used in AAL-5 encapsulation and PPP), or add a fixed value to the result (as in the ATM header CRC).

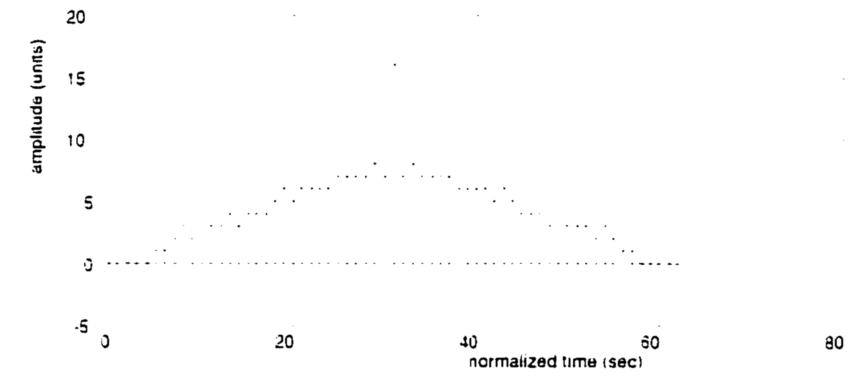


Figure 5.11 **Autocorrelation of DC balanced "Barker" sequence 0xB6AB31E0**

data interfering with the link. Given that self-synchronizing scramblers are considered an issue, it was decided to implement an independent scrambler.

5.3.7.1 Independent Scrambling Synchronization Design

Of the methods available, frame synchronization to the packet header is a bad idea because it provides a known sequence in a known location, which leaves it open for attack by hackers. The other technique, distributed sample scrambling, was another possibility, but was ruled out because of the following reasons:

1) Samples would have to be sent in the CRC: this would reduce the number of bits in the CRC that were known and would increase the probability of false frame markers. Note that from an error correction perspective, capabilities are not really weakened as five bits have to be removed in order to fall below double bit error correction capability.

2) The size of the packets is unknown, so GF(2) math operations would be required to properly set the received scrambler, based on the received samples. Other variants of this, such as sending the captured transmit scrambler state instead of samples lead to the same problem. This left set-reset scrambling as the only viable method of providing scrambling synchronization.

Given this result, the next step was to determine how to transmit the scrambler state at a known location, and select the length of the scrambler. These problems were attacked as follows:

1) In order to transmit the scrambler state, a special packet containing the scrambler state was designed. The first issue with this was to find a means of identifying it. Given a choice of packet length numbers between 0 and 65535, the obvious choices are either very small numbers or very large numbers. Influencing this decision are two factors:

a) IPv6 (Internet protocol version 6) may attempt to push for mega-packets in the network - up to 65535 bytes long[41].

b) One (and even two and three) byte packets are typically discarded as they are too small to carry any meaningful routing information in the network (i.e. they are really only useful in point-to-point systems).

c) One to three byte packets are typically filtered out as errors in most applications as protocols such as bit sync HDLC can produce one and two byte packets through a single bit error.

Consequently the value of one was chosen to represent a special packet used for scrambler state transmission, and the rule imposed that no packets of length one are allowed.

5.3.7.2 Scrambler Length Design

The next issue was the scrambler size. It was desired to select a scrambler with a long period to avoid any possibility of statistical scrambler attacks similar to the methods described for attacking POS in Appendix D. A scrambler size larger than 16 bits was considered adequate for this task, as this would eliminate the scrambler repeating during a maximum length packet.

The next consideration was that the scrambler length be a multiple of eight, as the protocol was targeted for byte oriented applications. The other consideration was that in high speed parallel applications circuitry would have to be designed to insert multiples of four byte gaps between non-contiguous packets for the null fill operation. Consequently, it was desired to make the special payload a multiple of four bytes in length.

The last factor that affects the choice of the length is that some measure of protection must be applied to this special payload to protect it from corruption. For this the same CRC-16 as used in the SDL header was selected, as it reduces the number of different CRC circuits that have to be designed in an implementation.

Given all of these considerations, a 6 byte scrambler was selected that used the primitive polynomial $g(x) = x^{48} + x^{28} + x^{27} + x + 1$.

5.3.7.3 Special Payload Protection

As mentioned above, a CRC-16 over the special payload was selected. Using eq. (5.1) and the knowledge that the CRC-16 polynomial was of the form $(x+1)p(x)$, it was found that the number of bits in the CRC field could theoretically afford double bit error correction over the 8 byte (48 + 16) payload[†]. However upon examining the minimum distance of the code, it was found that only single bit correction was capable of being supported[‡]. In reality this is not an issue, as the syndrome calculation circuitry required to handle double bit correction on a word of this size would be intolerable large. As well, the target bit error rate operation for most telecom links is in the vicinity of 10^{-9} with a signal degrade capability operation to 10^{-3}[7], so single bit error correction operation should be fine. This is shown as follows for a bit error rate of 10^{-9}:

$$\begin{aligned} P(>1\text{ error}) &= 1 - P(\text{zero errors}) - P(\text{one error}) \\ &= 1 - (1 - \text{BER})^{64} - 64\text{BER}(1 - \text{BER})^{63} \\ &= 4.03 \times 10^{-15} \end{aligned}$$

Similarly at BER = 10^{-3}:

† According to eq. (5.1) we would require at least 13 bits of redundancy.

‡ i.e. Some of the double bit error patterns produced syndromes that aliased into other error patterns.

$$\begin{aligned} P(>1 \text{ error}) &= 1 - P(\text{zero errors}) - P(\text{one error}) \\ &= 1 - (1 - \text{BER})^{64} - 64\text{BER}(1 - \text{BER})^{63} \\ &= 0.0019 \end{aligned}$$

This implies that even at the worst BER condition, the probability of receiving a multiply errored state is less than one in five hundred[†]. The result of this is that the scrambler state transmission packet looks as shown in Figure 5.12 (without the DC balance 0xB6AB31E0 added to the header):

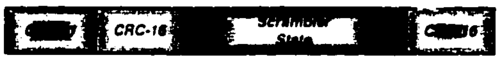


Figure 5.12 **Scrambler state packet**

5.3.7.4 Scrambling Exemptions

In order to allow synchronization to occur it is necessary to not scramble the SDL frame markers. As well, the special payload used to send the transmitter's scrambler state should not be scrambled as this is also part of the bootstrap process. This causes no problems as the state of the scrambler itself is random, and the most common SDL frame marker is the null fill, which has ideal properties (DC balanced and Barker-like).

5.3.8 OAM Messaging

The final consideration in protocol engineering is to consider whether to implement physical layer OAM (operation, administration and maintenance) messaging. The

[†] Note that this discussion is about whether it is worth trying to fix these packets, not if we detect the errors.

concept is to provide an out-of-band messaging channel to allow the two ends of the SDL link to communicate with each other. The reason for this is to allow the later implementation of software protocols associated with this link for things such as far end error reporting, in-service link parameter adjustments, etc.

After consideration, it was decided to implement a packet based messaging channel utilizing the same technique as the scrambler state transmission mechanism. This was accomplished by stealing the packet lengths 0x002 and 0x003. These lengths were selected using the same reasoning as was done in the scrambler state transmission. Namely that two and three byte packets do not really exist in a network. For instance, the minimum IP packet size is 40 bytes.

The result of this is that two special message packets are available, and these were designated as an "A" message and a "B" message. They are shown below (without DC balance and scrambling):

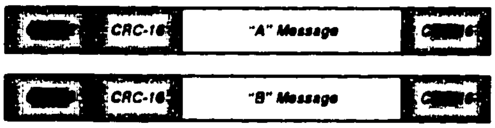


Figure 5.13 **"A" and "B" messages**

5.4 Estimated Performance

Given the protocol as specified, the overhead associated with this can now be calculated. If we assume an average packet size of 356 bytes, and a scrambler state retransmission time of 8 packets, the average link overhead is: $4 \times 8 + 12 = 44$

bytes per $8 \times 356 = 2848$ bytes transmitted. This results in a 1.5% inefficiency.

Comparing this with POS, we get:

Protocol	*Mapping inefficiency*
Byte Sync HDLC in SONET	**0.05**
IPv4 in SDL in SONET	**0.06**
IPv4 in SDL	**0.02**

Table 5.1 **Transport taxes associated with various formats**

As can be seen, IPv4 transported in SDL is very efficient.

SDL Performance 6-

6.1 Introduction

The purpose of this section is to investigate the performance of SDL with respect to various parameters such as packet length, packet mixture, BER, etc.

There are five key statistics that are very important in framing algorithms:

1) Mean time to frame (MTTF)

2) Mean time to synchronization (MTTS)

3) Probability of false frame (PFF)

4) Probability of false synchronization (PFS)

5) Probability of loss of frame (PLF)

This performance analysis will examine all five.

6.2 Mean Time to Frame (MTTF)

This parameter governs the length of time to frame from power up, as well as reframe times. Typically this is expressed in bytes or time, however since the SDL frame is directly tied to the packets it is transporting, it makes more sense to specify the reframe time in terms of packets. One of the reasons that this makes sense is that achieving frame synchronization in the middle of a packet is useless, as the entire packet is typically needed for information transmission.

The analysis version of the SDL framing state machine is shown in Figure 5.4, and the techniques used to compute MTTF are discussed in Section 5.3.3.3.

6.2.1 Results

The results for MTTF versus BER and number of framers are shown in Figures 5.6 - Figures 5.9. In Figures 6.1 - Figures 6.4, explicit graphs of the single and quad

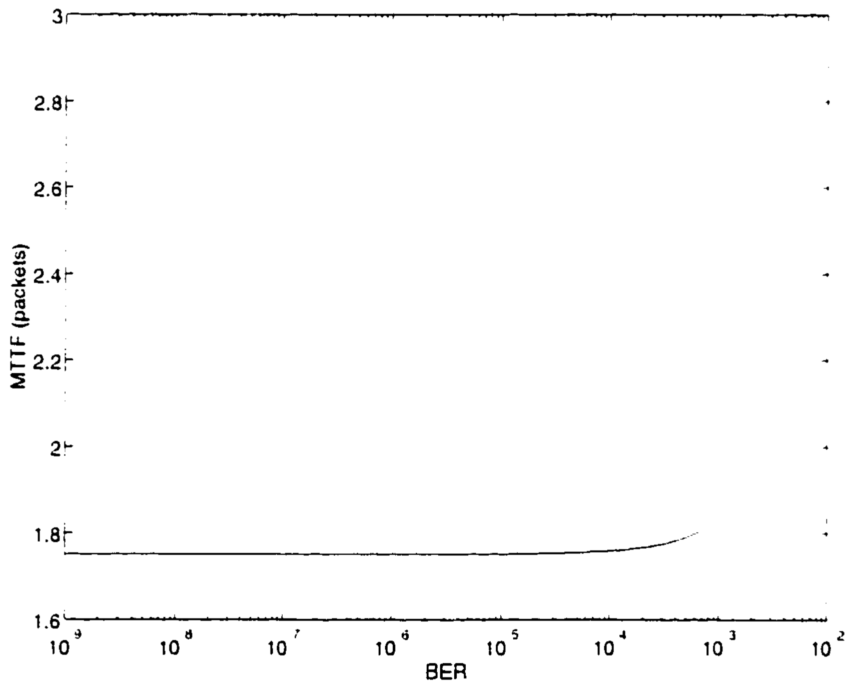


Figure 6.1 **Single framer MTTF versus BER at 356 byte average packet length**

scenarios are shown. As can be seen, 4 framers approaches the limit of a 1.5 packet MTTF for most bit error rates.

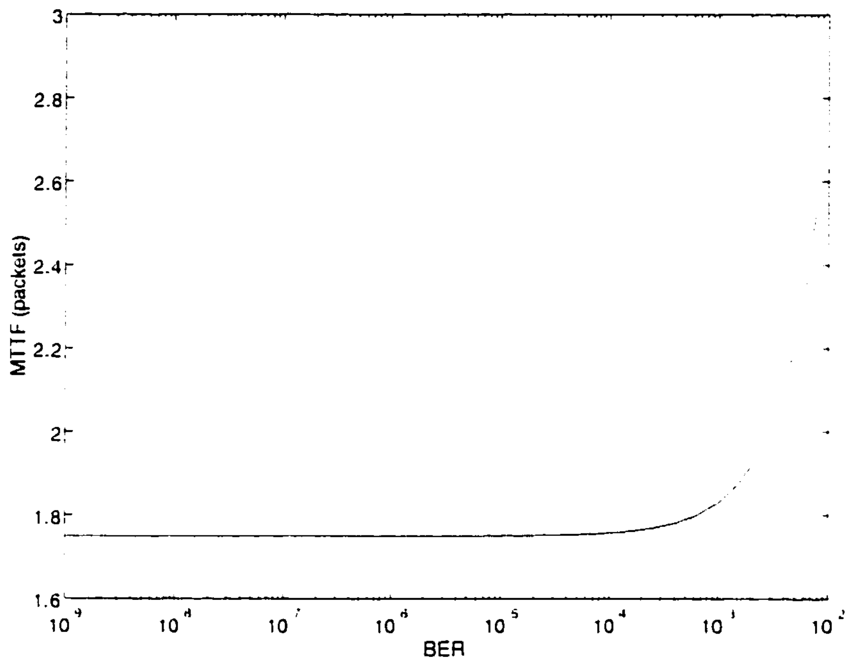


Figure 6.2 **Single framer MTTF versus BER at 64K byte average packet length**

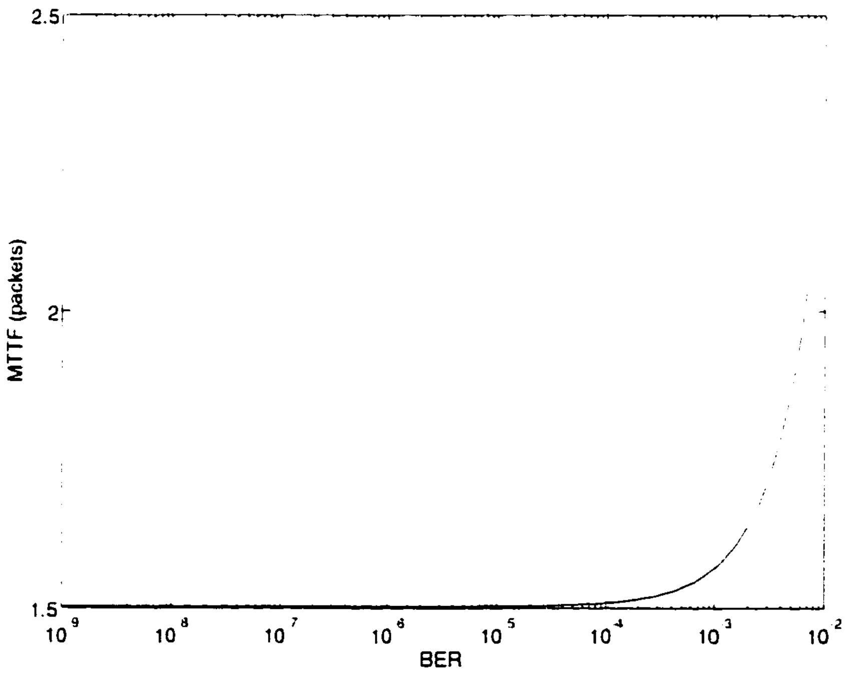


Figure 6.3 **Quad framer MTTF versus BER at 356 byte average packet length**

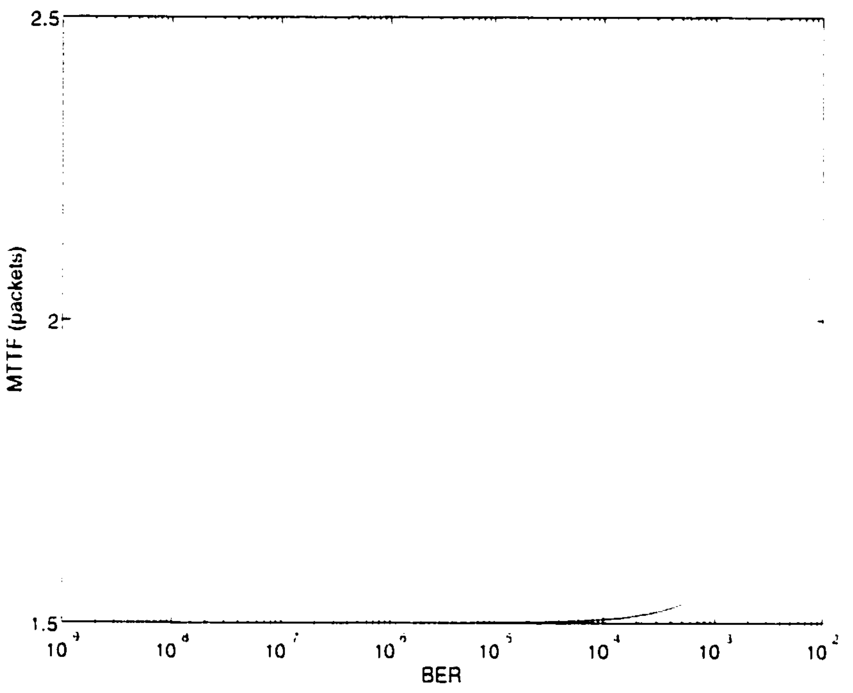


Figure 6.4 **Quad framer MTTF versus BER at 64K byte average packet length**

6.3 Mean Time to Synchronization (MTTS)

The mean time to synchronization will be the MTTF plus the state retransmit interval. For instance, if the state is transmitted every 8 packets, the limit of MTTS would be 5.5 packets. In order to receive a state transmission packet, both the header and the payload have to have less than one error in them. Thus:

$$p(\text{Miss State}) = p(\text{Header} > 1\ \text{Error}) \cdot p(\text{Payload} > 1\ \text{Error})$$
$$= (1 - (1 - \text{BER})^{32} - 32\text{BER}(1 - \text{BER})^{31}) \cdot (1 - (1 - \text{BER})^{64} - 64\text{BER}(1 - \text{BER})^{63}) \quad (6.1)$$

This probability is plotted in Figure 6.5. As can be seen, the probability is very low and inclusion of this effect in the MTTS is not necessary.

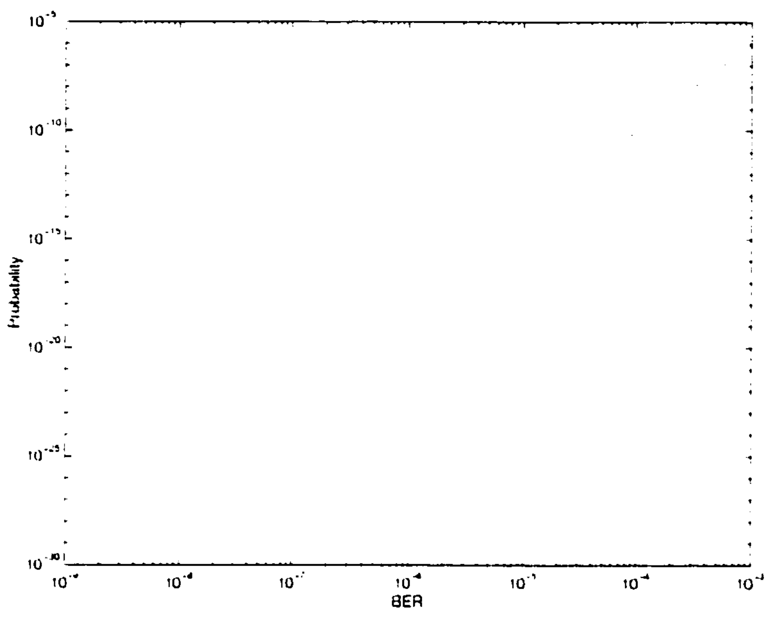


Figure 6.5 **Probability of missing synchronization packet versus BER**

6.4 Probability of False Frame (PFF)

The probability of false frame can be calculated straightforwardly as 2^{-32} or 232.8 x 10^{-12}. This is found from realizing that false framing requires 2 consecutive CRC-16 matches. A more detailed look would reveal some sensitivity to packet length, but the value of this is worthless when synchronization is considered, as this is required in order to pass data.

6.5 Probability of False Synchronization (PFS)

In order for false synchronization to occur, false frame is first required, followed immediately by a fake scrambler state packet (otherwise the probability drops by another 2^{-16})[†]. This would have to have a length field value of 0x0001, and a valid

CRC-16. This requires 4 events each with a probability of 2^{-16}. Thus the probability of false synchronization is $2^{-64} = 54.21 \times 10^{-21}$.

6.6 Probability of Loss of Frame (PLF)

PLF is directly a function of BER. Once synchronization is achieved, single bit header error correction is activated. Thus for loss of frame, more than one error in the header is required. This probability versus BER is plotted in Figure 6.6. For

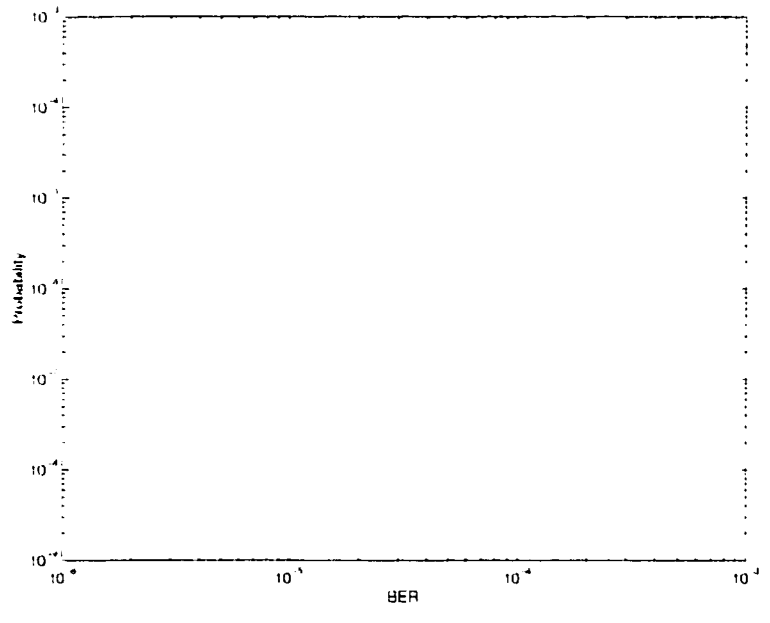


Figure 6.6 **Probability of Loss of Frame versus BER for single bit correction**

comparison's sake, a modified CRC-16 (using a primitive polynomial instead of the traditional 15th order polynomial multiplied by $x + 1$) could correct 2 errors in the

[†] The reasoning is that in order for false synchronization to be maintained, consecutive false frame markers must occur, each one with a probability of 2^{-16}.

32 bit header. The results for this are shown in Figure 6.7. As can be seen, the

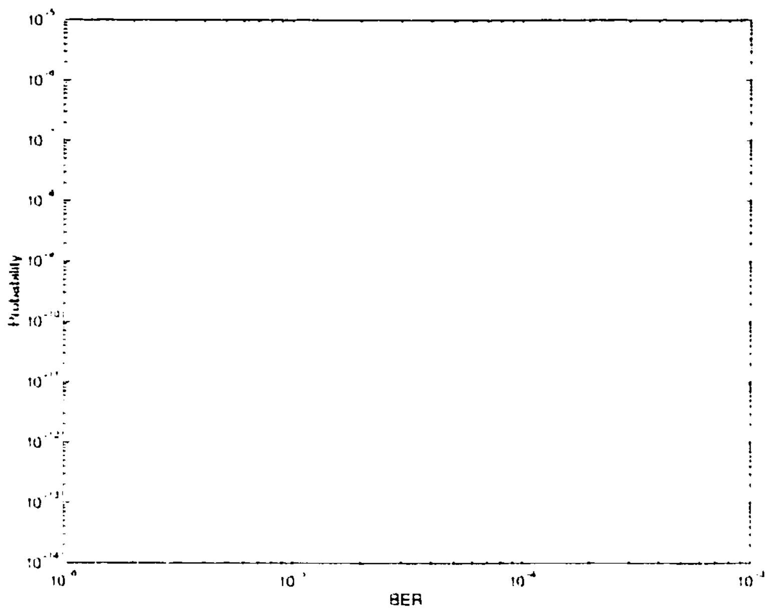


Figure 6.7 **Probability of Loss of Frame versus BER for double bit correction**

performance is significantly better at low values of BER, but at 10^{-3}, there is little gain.

Implementation 7-

7.1 Introduction

The purpose of this section is to provide an overview of the implementation of the framer.

7.2 Design Specification

The first implementation of the SDL protocol was targeted at an OC-48/STM-16 (2488 Mb/s) data transport chip (the TDAT042G5[†]). This chip was designed for "big-fat pipe" data termination applications, and for applications where a router hangs off of an ADM (add-drop multiplexer) on a ring[‡]. This chip was also intended to be a precursor to another chip with up to 48 channels (the TADM042G5). Because of this, it was desired to structure the architecture of the SDL transmit and receive blocks to handle an arbitrary number of channels. As well, the chips had to support multiprotocol operation (ATM and POS), which implied that the SDL portion of the chips had to be able to pass non-SDL data.

7.2.1 State Memory

The basic concept behind the framer is to use a context switching scheme to handle multiple channels in one framing device. This relies on having a memory such that

[†] See Appendix I for a TDAT042G5 product brief.

[‡] See Appendix D, Section D.1 for an explanation of these network topologies.

states can be addressed, read, and then updated: all within a single clock period. This concept is shown below:

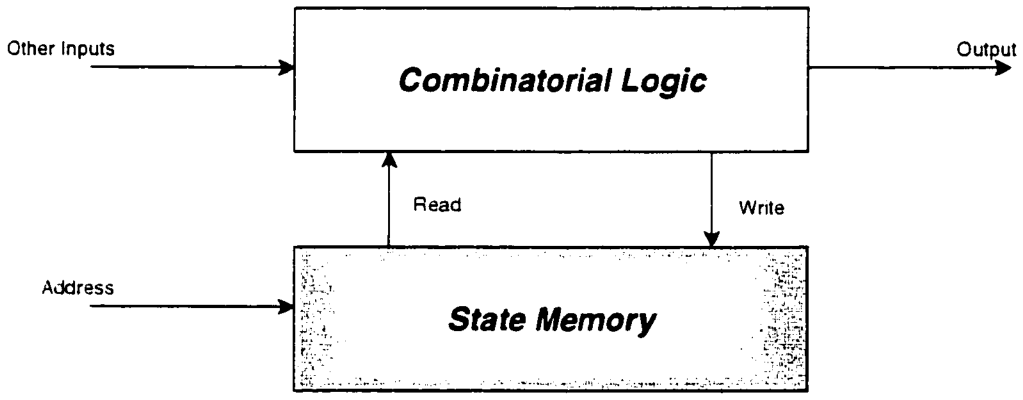


Figure 7.1 **Operation for a single clock cycle update**

Using this, it is possible to build the machinery that is independent of the number of channels, and if more channels are required only state memory needs to be added.

Based on this background, a 32 bit 78 MHz[†] data path was chosen[‡].

7.3 Data Engine Operation

7.3.1 Overview

The framers in the TDAT042G5 were constructed with the concept of having the data passed through the framer along with tags describing what it is, and consequently what operations should be performed upon it. The main signals being passed through each sub-block on a clock-by-clock basis are as follows:

[†] Actually 77.76 MHz.

[‡] In the actual design worst case slow (low voltage, and high temperature) and 80 MHz was used as a design target.

1) **ABCD** - 32 bits of data - all from the same channel, ordered in increasing time from the MSB of **A** to the LSB of **D**.

2) **ChannelID** - the channel number that **ABCD** belongs to.

3) **PayloadType** - the major class of payload associated with this particular channel (i.e. ATM, SDL, etc.).

4) **PayloadControl** - the details of this payload type (i.e. bit reversed CRC, etc.)

5) **PayloadMarker** - a flag for each byte in **ABCD** indicating whether this particular byte contains data

6) **EOPMarker, AbortMarker, DryMarker, BadMarker** - these are all EOP (end of packet) markers, and indicate an end of packet, and whether it is normal, aborted, dry, or bad. They are defined as unencoded to allow flexibility to expand the size of the data bus in the future for higher rates.

7) **Freeze** - used to pass blank bytes to the requesting block in the transmit direction. This allows for rate expansion caused by header insertion, etc.

8) **FreezeAddress** - the framer for which a freeze is destined.

Note that the channel settings for each block (such as PayloadType, etc.) could be independently stored in each sub-block (and indexed using the channel ID), but this would have duplicated the storage requirements in each block and would have complicated the register map.

7.3.2 Transmit Sequencing

When operating on a TDM scheduled SONET channel, the SDL framer will be used in conjunction with a transmit and receive sequencer, which will request particular channels from the UTOPIA+ packet interface (UTOPIA[51] with POS-PHY extension - a parallel packet interface with handshake signals) so that they arrive at the correct time for assembly into the SONET channel transmit payload[†]. This is shown in Figure 7.2.

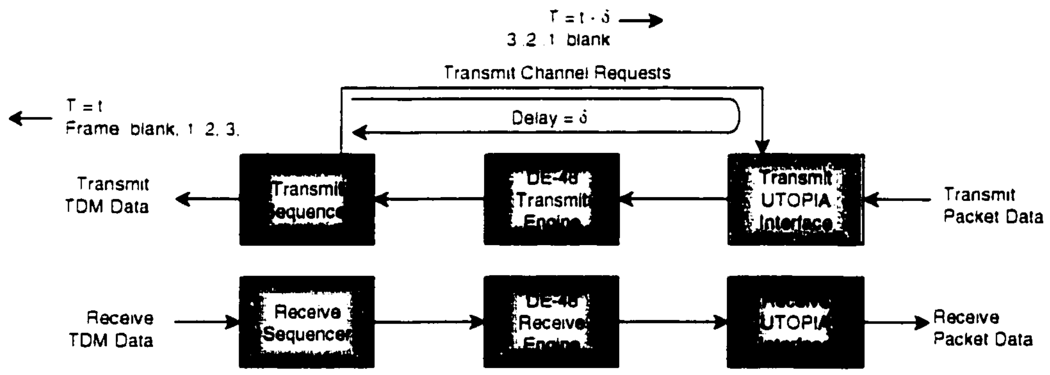


Figure 7.2 **Transmit sequencing**

7.4 Implementation Specifics

The block diagram of the system that was implemented is shown in Figure 7.3. These circuits were coded and simulated by the author in Verilog HDL and implemented in the Lucent Microelectronics 3.3V 0.25μ CMOS process in

[†] A patent was applied for on the details of this scheme.

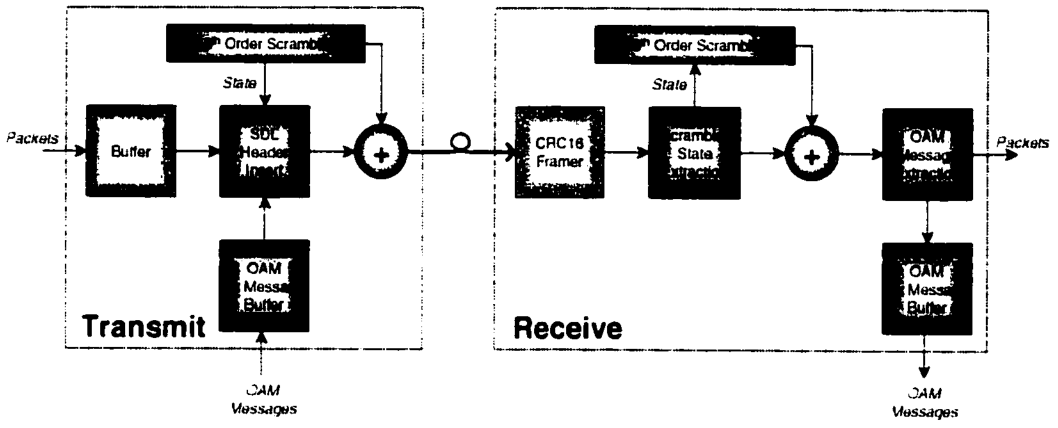


Figure 7.3 **Framer Tx/Rx block diagram**

Orlando, Florida. All back-end work (i.e. netlist, place and route, clock tree synthesis, scan insertion, parasitic extraction, back annotation, etc.) was done by the Lucent Microelectronics physical design engineering team in Allentown, PA.

7.4.1 Design Philosophy

The basic approach taken to designing the TDAT042G5 chip was to break the chip into major sections, with each major section then broken into individual blocks, each block owned by a designer. The SDL Tx/Rx blocks were considered individual blocks. Verification was done at the block level, major section level, and at the top level. The intent of the section and top level verification was to ensure that interconnectivity and control logic functionality was correct. Specific functional verification occurred only at the block level. The reason for this was to attempt to minimize the duration of the verification simulations, as full-chip simulation takes enormous amounts of computational power. Thus the role of the block designer was to produce functionally verified, synthesizable Verilog (or VHDL[†]), which was then handed off to the section team for integration and block level verification. To ensure margin at the top level, block level synthesis was performed at worst case slow conditions (high temperature, low voltage) at 100 MHz. Once the section was verified, this code was then handed off to the top level team for final verification and integration.

Upon completion of the top level verification, the code was handed off to the back end team which synthesized the code, and performed place and route, etc.[†]. Finally, the masks were cut, and the chip fabricated.

7.4.2 Validation Methodology

Chip validation[‡] was implemented using a combination of BIST, scan, and vectors. All memory structures implemented BIST (built in self-test), all synthesized CMOS flops were scannable (i.e. all flops were connected via serial chains that allowed the setting and reading of the values, thus providing a means of verifying the functionality of combinatorial logic), and at wafer probe, vectors were run. These vectors were created by the top-level verification team, and were chosen to roughly exercise the chip's major functions, hence providing a reasonable measure of the chip's "goodness" prior to packaging.

[*] A simulation environment capable of handling both was used.

[†] The back-end operation should not be viewed as trivial, as there are many long, involved operations in synthesizing a chip.

[‡] The act of verifying silicon is operational, versus verification which attempts to verify the correctness of the code.

7.5 SDL Frame Inserter

7.5.1 Interfaces

The interfaces for the SDL Frame Inserter are shown in Figure 7.4

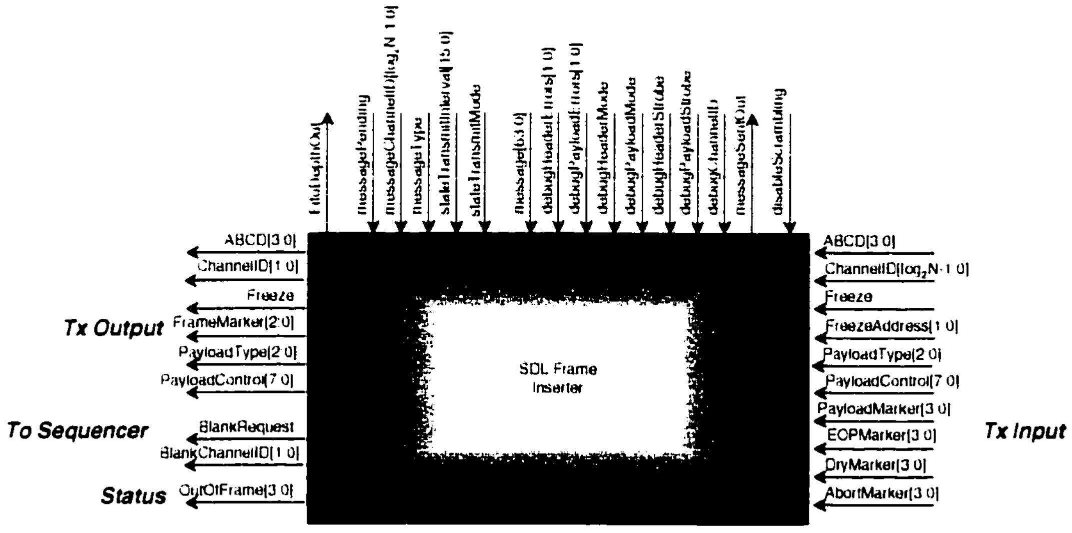


Figure 7.4 **SDL Frame Inserter Interfaces**

7.5.2 Transmit Input

The signals on this interface are listed inTable 7.1. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
A,B,C,D[7:0]	Input	These pins carry from one to four bytes of data for a channel with A[7:0] being the first byte received and D[7:0] being the last byte received, and the MSB of each byte being the first received (i.e. the right order for CRC calculation purposes).
ChannelID[log$_2$N-1:0]	Input	This signal indicates the logical channel ID of the data carried in ABCD.

Table 7.1 **Transmit input signal interface**

Signal Name	Direction	Description
Freeze	**Input**	Freeze = 1 indicates that four bytes of requested blanks are being passed in ABCD. Their destination is indicated by the FreezeAddress[1:0].
FreezeAddress[1:0]	**Input**	FreezeAddress is used in conjunction with the Freeze signal and indicates which sub-block requested the blanks. FreezeAddress[1:0] = 0x0: Gap Inserter FreezeAddress[1:0] = 0x1: HDLC Escaper FreezeAddress[1:0] = 0x2: SDL Frame Inserter FreezeAddress[1:0] = 0x3: Transmit Sequencer
PayloadType[2:0]	**Input**	This signal indicates the type of payload for the data carried in ABCD.
PayloadControl[7:0]	**Input**	This signal carries the mode control bits for the data carried in ABCD.
PayloadMarker[3:0]	**Input**	These flags are used to indicate which (if any) of the four bytes is blank. PayloadMarker[3] corresponds to the byte in A[7:0], PayloadMarker[2] corresponds to B[7:0] etc. PayloadMarker: 1 = payload, 0 = undefined
EOPMarker[3:0]	**Input**	These bits indicate where an EOP occurs, with EOPMarker[3] corresponding to the byte in A[7:0], etc. Note that a byte can only be flagged as either EOP, Dry, or Abort, not both.
DryMarker[3:0]	**Input**	These bits indicate where a FIFO has gone dry, with DryMarker[3] corresponding to the byte in A[7:0], etc. Note that a byte can only be flagged as either EOP, Dry, or Abort, not both.
AbortMarker[3:0]	**Input**	These bits indicate where an abort event has occurred, with AbortMarker[3] corresponding to the byte in A[7:0], etc. Note that a byte can only be flagged as either EOP, Dry, or Abort, not both.

Table 7.1 **Transmit input signal interface**

7.5.3 Transmit Output

The signals on this interface are listed in Table 7.2. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
A,B,C,D[7:0]	**Output**	These pins carry from one to four bytes of data for a channel with A[7:0] being the first byte received and D[7:0] being the last byte received, and the MSB of each byte being the first received (i.e. the right order for CRC calculation purposes).
ChannelID[$\log_2 N$-1:0]	**Output**	This signal indicates the logical channel ID of the data carried in ABCD.
Freeze	**Output**	Freeze = 1 indicates that four bytes of requested blanks for the transmit sequencer are being passed in ABCD.
PayloadType[2:0]	**Output**	This signal indicates the type of payload for the data carried in ABCD.
PayloadControl[7:0]	**Output**	This signal carries the mode control bits for the data carried in ABCD.
FrameMarker[2:0]	**Output**	These bits indicate where the last byte in the SONET/SDH frame is. It is used in transparent mode to align the payload to the SONET/SDH frame. FrameMarker[2] = 1 indicates the presence of an end-of-frame. If FrameMarker[2] = 1, then FrameMarker[1:0] indicates where within ABCD the end-of-frame lies. 0x00 corresponds to A[7:0] and 0x03 corresponds to D[7:0].

Table 7.2 **Transmit output signal interface**

7.5.4 Status Output

The signals on this interface are listed in Table 7.3. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
OutOfFrame[N-1:0]	**Output**	These pins indicate whether the ATM framer and scrambler is in the sync state or not.

Table 7.3 **Pointer Interpreter signal interface**

7.5.5 Sequencer Output

The signals on this interface are listed inTable 7.4. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
BlankRequest	**Output**	BlankRequest is used in conjunction with BlankChannelID to request that the next time the channel indicated in BlankChannelID is requested, it be blanked (i.e. a block in the data engine has an extra word of data for this channel).
BlankChannelID[1:0]	**Output**	BlankChannelID indicates the channel number that BlankRequest is referring to.

Table 7.4 **Sequencer transmit signal interface**

7.5.6 Control Interface

The signals on this interface are listed inTable 7.4. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
FIFODepthOut	**Output**	FIFODepth out is used as a diagnostic signal to indicate whether the FIFO in the module SDL_Tx_Buffer is above or below the halfway mark. When greater than half full for the current channel (as indicated by the input channelID), FIFODepthOut = 1.
messagePending	**Input**	messagePending is used to indicate to the module SDL_Tx_Machine that either an "A" or "B" message is available for transmission. When a message is pending, messagePending = 1.
messageChannelID [log$_2$N-1:0]	**Input**	messageChannelID indicates the logical channel that a message (as indicated by messagePending) is to be sent on.

messageType	**Input**	messageType indicates the type of message (as indicated by messagePending) that is to be sent.
		messageType = 0: A Message
		messageType = 1: B Message

Table 7.5 **Control signal interface**

Signal Name	Direction	Description
message[63:0]	**Input**	message contains a prepared message consisting of 6 bytes of data and 2 bytes of CRC-16 calculated over the data.
messageSentOut	**Output**	messageSentOut = 1 indicates that the message (as indicated by messagePending) has been sent.
stateTransmitInterval [15:0]	**Input**	stateTransmitInterval indicates the number of D-Words or packets (depending on stateTransmitMode) that must pass between sending the scrambler state
stateTransmitMode	**Input**	stateTransmitMode indicates whether the interval between consecutive scrambler state transmissions is measured in D-Words or packets

debugHeaderErrors [1:0]	**Input**	debugHeaderErrors indicates the number of errors to insert in the SDL header on a given channel (as specified by debugChannelID) for debug purposes. The error injection is done using a walking ones pattern to cover all possibilities.
		
		When no error injection is desired, this should be set to 0x0.
		
		debugHeaderErrors = 0x0: 0
		debugHeaderErrors = 0x1: 1
		debugHeaderErrors = 0x2, 0x3: 2

debugPayloadErrors [1:0]	**Input**	debugPayloadErrors indicates the number of errors to insert in the payload of special packets on a given channel (as specified by debugChannelID) for debug purposes. The error injection is done using a walking ones pattern to cover all possibilities.
		
		When no error injection is desired, this should be set to 0x0.
		
		debugPayloadErrors = 0x0: 0
		debugPayloadErrors = 0x1: 1
		debugPayloadErrors = 0x2, 0x3: 2

debugHeaderMode	**Input**	debugHeaderMode indicates whether the header error injection functions in continuous or single shot.
		
		debugHeaderMode = 0: Continuous
		debugHeaderMode = 1: Single Shot

debugPayloadMode	**Input**	debugPayloadMode indicates whether the payload error injection functions in continuous or single shot.
		
		debugPayloadMode = 0: Continuous
		debugPayloadMode = 1: Single Shot

Table 7.5 **Control signal interface**

Signal Name	Direction	Description
debugHeaderStrobe	**Input**	debugHeaderStrobe is used to initiate a single shot injection of header errors when in Single Shot mode as specified by debugHeaderMode. Upon transitioning this input from 0 to 1, errors (as specified by debugHeaderErrors) will be injected into the next available header.
debugPayloadStrobe	**Input**	debugPayloadStrobe is used to initiate a single shot injection of header errors when in Single Shot mode as specified by debugPayloadMode. Upon transitioning this input from 0 to 1, errors (as specified by debugPayloadErrors) will be injected into the next available special payload.
debugChannelID [log$_2$N-1:0]	**Input**	debugChannelD specifies the ID of the channel into which errors shall be injected for debug purposes.
disableScrambling	**Input**	disableScrambling = 1 inhibits the scrambling of the data stream. All scrambler machinery (i.e. state transmissions) continues operation in a normal fashion - only the data stream is not scrambled. This is useful for lab debugging purposes.

Table 7.5 **Control signal interface**

7.5.7 SDL Frame Inserter Details

The functional implementation of the SDL frame inserter is shown in Figure 7.5, and was implemented using the blocks shown in Figure 7.6.

7.5.7.1 SDL Tx Buffer

The purpose of the SDL Tx Buffer is to buffer D-Words while special payloads are being inserted into the data stream. Whenever there is data in the FIFO and something is written and nothing read, a blank must be requested. As well, if a blank shows up, and nothing was read, another blank must be requested, as the purpose of blanks is to drop the FIFO fill by one if nothing is read (i.e. two back to back special payloads, etc.). This handshake is accomplished through the dataAvailableOut and

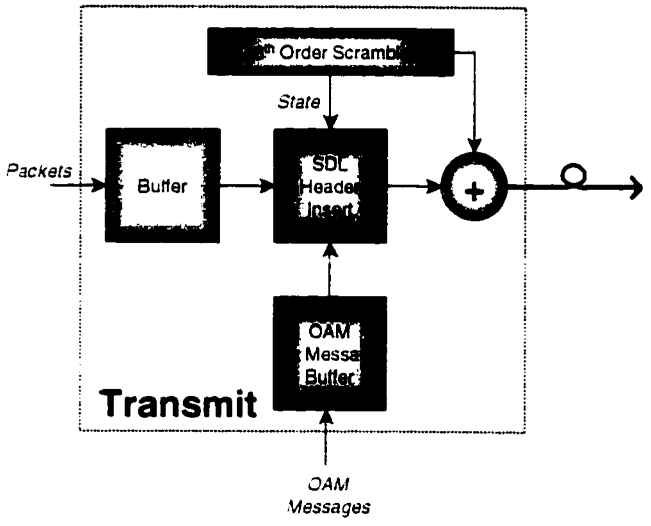


Figure 7.5 **Functional implementation**

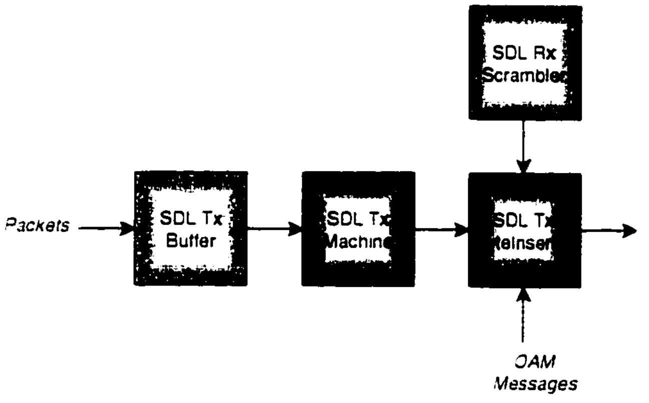


Figure 7.6 **Actual implementation**

readData signals. The only time that a blank is not requested is when the block is powered up and the FIFO is empty. In this case, the SDL Tx Buffer will store the first D-Word before passing data. This produces a net delay through this block of one D-Word. Note that all incoming data is written into the FIFO including interpacket gaps. One may be tempted to use intrapacket gaps the same as blanks. However this produces a pathological situation involving requesting blanks, and then having some naturally occurring interpacket gaps show up, each dropping the

FIFO depth by one. Then, when a new packet starts, the blanks that were requested earlier arrive causing FIFO exhaustion.

To assist in the debugging of this block in the lab, a signal called FIFODepthOut is provided that indicates whether the FIFO is greater than half full.

This block is designed to interface directly to the SDL Tx Machine, and as such does not have latched outputs.

The control signals associated with this block are:

1) dataAvailableOut - when set to one this indicates that there is data to be read in the FIFO

2) FIFODepthOut - when this is one it indicates that the FIFO is over half full

3) readData - this is an input from the SDL Tx Machine indicating that a D-Word was read from the FIFO

7.5.7.2 SDL Tx Machine

The SDL Tx Machine sub-block functions as a classifier for the four bytes of data in abcd. Each byte is assigned a header type (hTa -> hTd) and a header count (hCa -> hCd). The legitimate values of the header type:

0x0 = null fill
0x1 = scrambler state
0x2 = A message

0x3 = B message
0x4 = packet

The legitimate values of the header count are 0x0 -> 0xC, and correspond to what byte in the header this is

Header Type	Header Count Range
0x0	0x0 -> 0x3
0x1	0x0 -> 0xB
0x2	0x0 -> 0xB
0x3	0x0 -> 0xB
0x4	0x0 -> 0x3, 0xC

Table 7.6 **Allowed header counts versus header type**

For instance, the null fill header only has four bytes in it, while the special payloads (0x0001, 0x0002, and 0x0003) have 12 bytes: the first four being the SDL frame marker, the next six being the special payload, and the 11th and 12th being the special payload CRC. Finally for packets, the first four bytes of the SDL header are labelled 0x0 -> 0x3, whereas the payload is labelled 0xC.

Thus the SDL header is always contained in bytes whose header count is between 0x0 and 0x3.

The SDL Tx Machine resets itself always based on the end of packet marker, and takes as its input a signal called messagePending which indicates that a message is available for sending. This block also contains the D-Word or packet counter for monitoring when to send special packets.

The control outputs from this block are:

1) hTa -> hTd - these indicate the header type of the byte corresponding to a, b, c, or d

2) hCa -> hCd - these indicate the header count of the byte corresponding to a, b, c, or d

3) scramblerStateRequired - when set to 1 this signal indicates to the SDL Tx Scrambler that a scrambler state is required

4) scramblerStateOffset[1:0] - this signal indicates the required offset based on where the beginning of the scrambler state in the D-Word (a, b, c, or d) is. 0x0 indicates that the special payload will start in byte a

7.5.7.3 SDL Tx Scrambler

This block contains the x^{48} scrambler, and the circuitry needed to implement the scrambler state offset adjustments (to compensate for the differing positions that the scrambler state may need to be transmitted at).

This block also outputs the scrambler values for the SDL Tx ByteInserter so that the data stream can be scrambled.

The control signals associated with this subblock are:

1) scramblerState[63:0] - this output contains the scrambler state with the correct offset, and the CRC16 attached

2) scramblerOutput[31:0] - this contains the scrambler output

7.5.7.4 SDL Tx ByteInserter

This block is responsible for the final assembly of the outgoing data stream. It receives all of the hT and hC tags for the bytes, as well as the scrambler states, scrambler output values, and messages to accomplish this task.

To enable debugging, this block contains circuitry to inject a marching error sequence into the headers and special payloads, as well as single shot errors. The purpose of this is to enable debugging of the error counters in the lab.

The control signals associates with this subblock are:

1) messageSentOut - when set to one, this indicates that the pending message has been sent

7.6 SDL Framer

7.6.1 Interfaces

The interfaces for the SDL Framer are shown in Figure 7.4

7.6.1.1 Receive Input

The signals on this interface are listed in Table 7.7. All signals are valid on the rising edge of the clock.

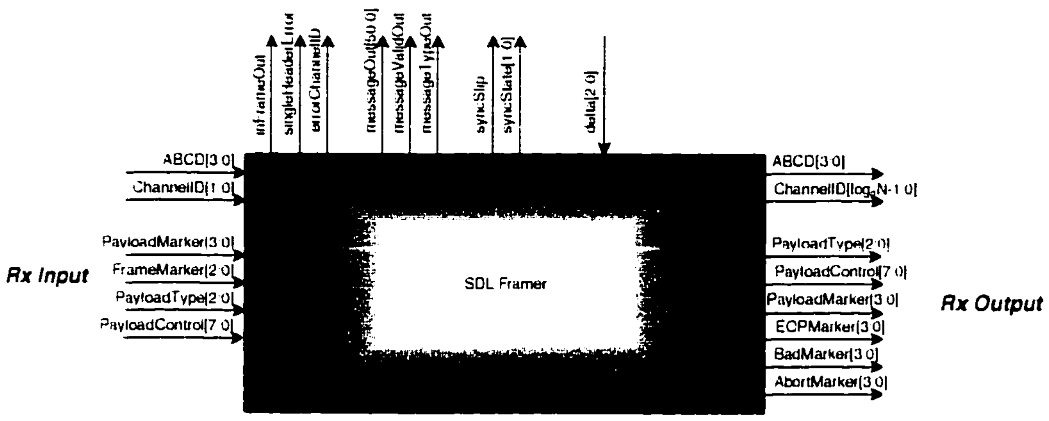


Figure 7.7 **SDL Framer Interfaces**

Signal Name	Direction	Description
A,B,C,D[7:0]	**Input**	These pins carry from one to four bytes of data for a channel with A[7:0] being the first byte received and D[7:0] being the last byte received, and the MSB of each byte being the first received (i.e. the right order for CRC calculation purposes).
ChannelID[log$_2$N-1:0]	**Input**	This signal indicates the logical channel ID of the data carried in ABCD.
PayloadType[2:0]	**Input**	This signal indicates the type of payload for the data carried in ABCD.
PayloadControl[7:0]	**Input**	This signal carries the mode control bits for the data carried in ABCD.
PayloadMarker[3:0]	**Input**	These flags are used to indicate which (if any) of the four bytes is blank. PayloadMarker[3] corresponds to the byte in A[7:0], PayloadMarker[2] corresponds to B[7:0] etc. PayloadMarker: 1 = payload, 0 = undefined

Table 7.7 **Receive input signal interface**

7.6.1.2 Receive Output

The signals on this interface are listed in Table 7.8. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
A,B,C,D[7:0]	**Output**	These pins carry from one to four bytes of data for a channel with A[7:0] being the first byte received and D[7:0] being the last byte received, and the MSB of each byte being the first received (i.e. the right order for CRC calculation purposes).
ChannelID[$\log_2 N$-1:0]	**Output**	This signal indicates the logical channel ID of the data carried in ABCD.
PayloadType[2:0]	**Output**	This signal indicates the type of payload for the data carried in ABCD.
PayloadControl[7:0]	**Output**	This signal carries the mode control bits for the data carried in ABCD.
PayloadMarker[3:0]	**Output**	These flags are used to indicate which (if any) of the four bytes is blank. PayloadMarker[3] corresponds to the byte in A[7:0], PayloadMarker[2] corresponds to B[7:0] etc. PayloadMarker: 1 = payload, 0 = undefined
EOPMarker[3:0]	**Output**	These bits indicate where an EOP occurs, with EOPMarker[3] corresponding to the byte in A[7:0], etc. Note that a byte can only be flagged as either EOP, or Abort, not both.

Table 7.8 **Receive output signal interface**

7.6.1.3 Control Interface

The signals on this interface are listed inTable 7.9. All signals are valid on the rising edge of the clock.

Signal Name	Direction	Description
inFrameOut	**Output**	inFrameOut = 1 indicates that the framer on the current channel (as indicated by the output channelID) is in frame.
singleHeaderError	**Output**	singleHeaderError = 1 indicates whether a correctable error occurred on the channel indicated by errorChannelID.
errorChannelID [$\log_2 N$-1:0]	**Output**	errorChannelID indicates the channel ID on which an error (as reported by singleHeaderError) occurred.

Table 7.9 **Control signal interface**

Signal Name	Direction	Description
messageOut[50:0]	**Output**	messageOut[47:0] contains a received, and if possible, corrected message. messageOut[49:48] contains the number of errors in the message: messageOut[49:48] = 0x0: no errors messageOut[49:48] = 0x1: 1 errors messageOut[49:48] = 0x2, 0x3: 2 or more errors messageOut[50] = 1 indicates if the message is valid (i.e. whether there were zero or one errors in the payload).
messageValidOut	**Output**	messageValidOut = 1 indicates that a received message is contained in messageOut for the channel specified by the output channelID.

messageTypeOut	**Output**	messageTypeOut indicates the type of the received message:
		messageTypeOut = 0: A Message
		messageTypeOut = 1: B Message

syncSlip	**Output**	syncSlip = 1 indicates that a synchronization slip (see the SDL spec) has occurred on the channel indicated by the output channelID
syncState[1:0]	**Output**	syncSlip = 1 indicates that a synchronization slip (see the SDL spec) has occurred on the channel indicated by the output channelID
delat[2:0]	**Input**	delta controls the number correct frame markers the framer must see before declaring frame synchronization

Table 7.9 **Control signal interface**

7.6.2 SDL Framer Details

The functional implementation of the SDL framer is shown in Figure 7.8, and was implemented using the blocks shown in Figure 7.9.

7.6.2.1 SDL Rx Framer

The CRC16 framer searches all legitimate bit positions (i.e. every eight if byte aligned, versus every bit if bit aligned) for 32 bit combinations that could be SDL

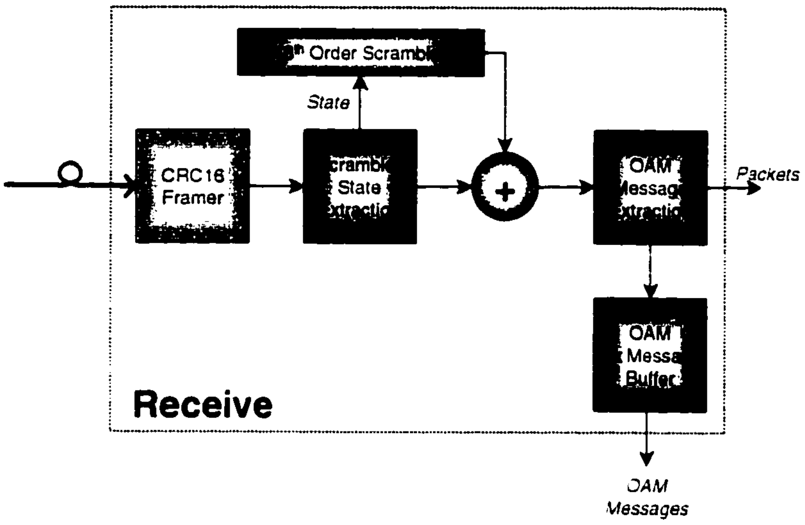


Figure 7.8 **Functional implementation**

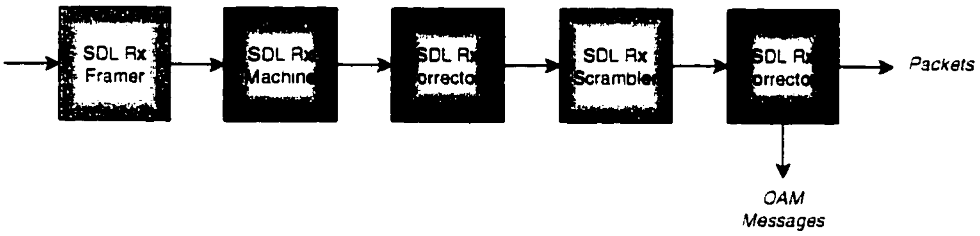


Figure 7.9 **Actual implementation**

headers. This is done by 32 parallel CRC16 checkers, and the output is one-hot encoded into a 32 bit "hitMask". Since there can be at most one legitimate frame position within a 32 bit boundary, the encoding arbitrarily selects the first occurrence in time (i.e. bit 31 is the first and bit 0 is the last). This hit mask is then fed to four parallel hunt framer circuits whose job it is to track the hits in the hit mask to see if we have found frame. These framers are merely counters and have no error correcting capability. Their operation is as follows:

1) The hunt framers are essentially in a "free FIFO".

2) Upon the first occurrence of a "hit", the first free hunt framer is sent to chase the occurrence.

3) Subsequent "hits" are assigned to the other free hunt framing circuits.

4) This operation continues until one of the active hunt framer declares synchronization, in which case all of the other hunt framers stop, and the active hunt framer passes it's position to a real framer which has the capability of correcting single bit errors in the header.

5) In order to prevent multiple hunt framers from tracking the same occurrence (i.e. if δ is ever set to greater than two) there is a "kicker" circuit which inactivates a hunt framer which attempts to track a header which is already being tracked.

In order to meet timing on this circuit, pipelining of the operation was required.

The control outputs from this block are:

1) singleBitError - indicates that a single bit was corrected in an SDL header.

2) inFrame - indicates that the we are in frame.

3) eop[3:0] - a one-hot encoded end of packet marker indicating the last byte of the packet.

4) nextHeaderType[2:0] - this is indicates that the next SDL packet type will be and is set when eop is non-zero.

0x0 = null fill
0x1 = scrambler state
0x2 = A message
0x3 = B message
0x4 -> 0x7 = packet

7.6.2.2 SDL Rx Machine

The SDL Rx Machine sub-block functions as a classifier for the four bytes of data in abcd. Each byte is assigned a header type (hTa -> hTd) and a header count (hCa -> hCd). The legitimate values of the header type:

0x0 = null fill
0x1 = scrambler state
0x2 = A message
0x3 = B message
0x4 = packet

The legitimate values of the header count are 0x0 -> 0xC, and correspond to what byte in the header this is

Header Type	Header Count Range
0x0	0x0 -> 0x3
0x1	0x0 -> 0xB
0x2	0x0 -> 0xB
0x3	0x0 -> 0xB
0x4	0x0 -> 0x3, 0xC

Table 7.10 **Allowed header counts versus header type**

For instance, the null fill header only has four bytes in it, while the special payloads (0x0001, 0x0002, and 0x0003) have 12 bytes: the first four being the SDL frame marker, the next six being the special payload, and the 11th and 12th being the special payload CRC. Finally for packets, the first four bytes of the SDL header are labelled 0x0 ->0x3, whereas the payload is labelled 0xC.

Thus the SDL header is always contained in bytes whose header count is between 0x0 and 0x3.

The control outputs from this block are:

1) hTa -> hTd - these indicate the header type of the byte corresponding to a, b, c, or d.

2) hCa -> hCd - these indicate the header count of the byte corresponding to a, b, c, or d.

7.6.2.3 SDL Rx Corrector1

The purpose of this block is to extract and correct received scrambler states. This is possible since the special payload of the scrambler state is not scrambled. These extracted states are forwarded to the next sub-block which is the SDL Rx Scrambler.

The control outputs from this block are:

1) scramblerStateOut[50:0] - this contains the corrected scrambler state in the first 48 bits, the next two bits contain the number of errors found (0, 1, or 2 or more), and the last bit indicates whether the special payload is usable (i.e. zero or one error).

2) scramblerOffsetOut[1:0] - this indicates which byte (a, b, c, or d) contained the start of the special payload. This is required in order to properly synchronize the scrambler. 0x0 corresponds to byte a.

3) scramblerStateValidOut - this indicates to the SDL Rx Scrambler that a new scrambler state has been received.

7.6.2.4 SDL Rx Scrambler

This block contains the x^{48} descrambler, and the circuitry needed to implement the incoming payload offset adjustments (to compensate for the differing positions that the scrambler state may arrive at). This offset adjustment takes into account all of the delays associated with the transmittal of the state.

This block also performs the unscrambling operation on the data passing through it (contained in abcd).

The control outputs from this block are:

1) syncStateOut[1:0] - this indicates what state the scrambler for this channel is in. A value of 0x0 corresponds to unsynchronized, 0x1 to synchronized, and 0x2 to post-synchronized (i.e. a scrambler state arrived that did not match the current state).

2) syncSlipOut - this indicates whether a sync slip occurred (i.e. if 2 consecutive scrambler states arrive that do not match the received scrambler's state, the receive scrambler is resynchronized to the second differing state, and a sync slip is declared by setting syncSlip to 1).

7.6.2.5 SDL Rx Corrector2

The purpose of this block is to extract and correct received messages, and to set the correct payload marker for the output data.

The control outputs from this block are:

1) messageOut[50:0] - this contains the corrected message in the first 48 bits, the next two bits contain the number of errors found (0, 1, or 2 or more), and the last bit indicates whether the special payload is usable (i.e. zero or one error).

2) messageTypeOut - this indicates the type of the received message. 0 corresponds to A, and 1 to B.

3) messageValidOut - this indicates that a new message has been received.

Results 8-

8.1 Introduction

The purpose of this section is to present the results of the framer implementation described in Chapter 7.

8.2 Die

Figure 8.1 shows a photograph of a layout plot of the TDAT042G5. It was

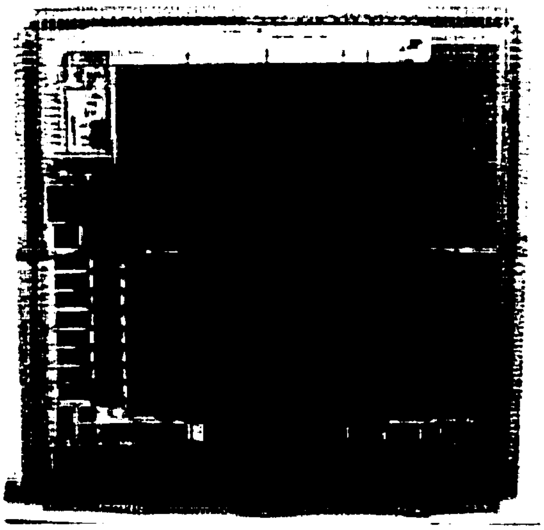


Figure 8.1 **TDAT042G5 die layout**

implemented in Lucent Microelectronic's 0.25μ CMOS process, and has 600 I/Os. The first dense section below the equator of the die is the data framer section where the SDL framer was implemented.

8.3 Package

Figure 8.2 shows a photograph of the TDAT042G5 and a lower speed version of the

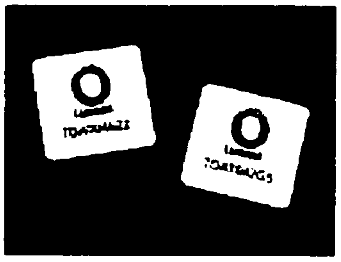


Figure 8.2 **Packaged TDAT042G5 and lower rate version**

same die. It was packaged in a 600 LBGA (laminated ball grid array) package.

8.4 Test Board

Figure 8.3 shows the TDAT042G5 OC-48 test board which was used to demonstrate

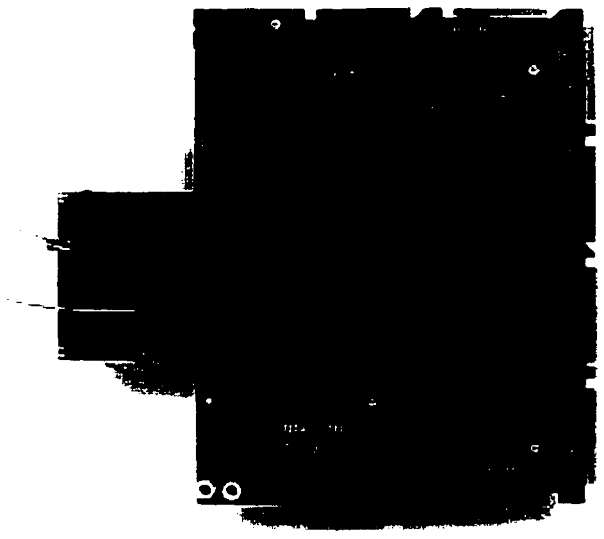


Figure 8.3 **TDAT042G5 OC-48 test board**

(for the first time) 2488 Mb/s packet directly over fiber using SDL. This first was done on May 6, 1999 in the Lucent ME lab in Murray Hill, NJ by Tom Egolf and Ut-Va Koc. The test set is shown in Figure 8.4, and was used for all of the work at

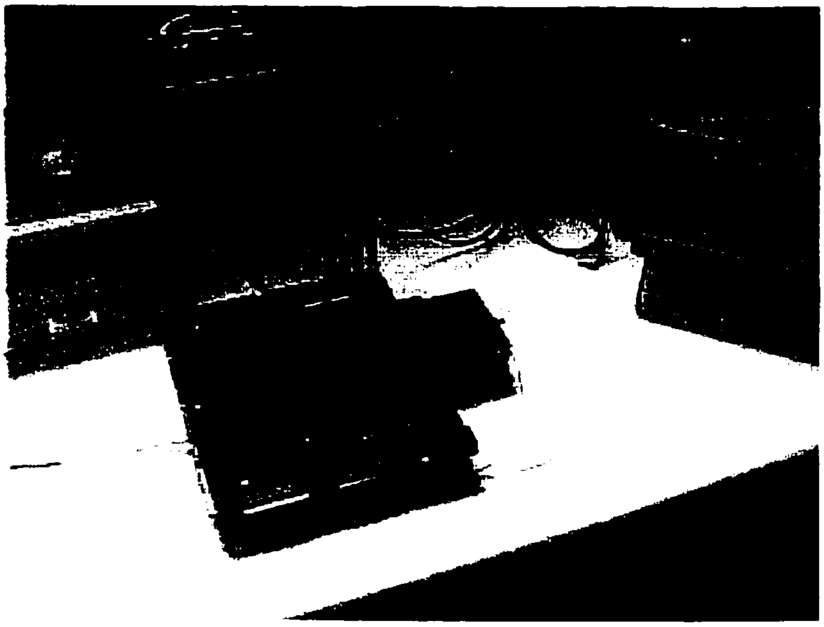


Figure 8.4 **TDAT042G5 OC-48 test set**

2488 Mb/s, whether in SONET/SDH or directly over fiber.

8.5 Results

As expected of any digital circuit that has been properly verified, the SDL protocol (and the TDAT042G5) functioned exactly as intended. Up to this point in time, extensive testing has been completed of the SDL protocol over long periods of operation (days on end error free), both in SONET/SDH payloads, and directly over fiber at 155 Mb/s, 622 Mb/s, and 2488 Mb/s. In operation the TDAT042G5 at 2488 Mb/s operation consumed 6.5 W with the data path running 32 bits at 78 MHz.

Summary and Suggestions for Future Work 9-

9.1 Introduction

The purpose of this section is to review the current state of SDL and suggest future directions for work.

9.2 Summary

In this thesis the development of a robust, efficient physical layer transport protocol for packets was discussed. All of the theory and numerical techniques to develop and analyze the performance of this protocol have been developed, and the results of this analysis under a variety of environments presented. An analysis and discussion of the shortcomings of the current state of the art (POS) was also presented, allowing a contrast between the two methods to be made.

Finally, an implementation of this framer in an integrated circuit running at 2488 Mb/s was detailed, and the results in the lab discussed. Issues concerning the implementation at this high speed were also discussed.

Below, the current state of SDL in the various standards bodies is summarized and suggestions for future work given.

9.3 Standards Status

9.3.1 IETF

The IETF (Internet Engineering Task Force) is the first standards forum where SDL was presented. Currently SDL is being very well received after an initial battle over the purported weaknesses in POS. Having convinced the IETF PPP Extensions work group (the group that handles POS and SDL) over a 2 month period (during which the results of Appendix D were presented to the e-mail exploder) of the flaws in the RFC1619 POS standard (especially for clear channel mappings). SDL is on its way to being approved and receiving RFC status. The draft submitted to the June 1999 meeting is given in Appendix J. As of this meeting. SDL was granted standards status as soon as the ITU grants C2 path signal labels for SDL. SInce this was achieved at the November meeting of the ITU T1X1 committee. it is expected that SDL will be a standard with the next couple of months.

As well. SDL is being standardized for packet directly over fiber operation. and the first draft of this standard is given in Appendix K.

9.3.2 ITU

SDL has been recognized by the ITU in the November 1999 meeting of the T1X1 committee. where it was given C2 path signal labels. This will appear in the next issue of G.707. As well. Nortel has adopted SDL for transporting Gigabit Ethernet over SONET/SDH and is proposing this in the ITU as a standard.

9.3.3 OIF

The Optical Internetworking Forum was the second standards body where SDL was presented. It currently has an experimental status there, and will probably remain in this position until it becomes an IETF RFC. This standards body is extremely political, and SDL is being fought by companies that currently do not have an implementation.

9.3.4 10G Ethernet

SDL has recently been proposed (March 1999) to the 10G Ethernet standards committee, and is one of two competing proposals. The second uses 8B/10B coding and is far less appealing than SDL as the bit rate increases to 12.5 Gb/s. As a result SDL appears likely to become the standard for 10G Ethernet.

9.3.5 System I/O

System I/O is a standard that is being developed for the replacement of the PCI bus in the form of a 2.5 Gb/s serial link. SDL is being considered for over-fiber operation on long links. This standard is being supported by 3Com, Sun and Intel, Compaq, HP, Lucent, and IBM. It is the amalgamation of the NGIO and Future I/O efforts.

9.4 Future Work

The most immediate work that is required on SDL is the definition of the OAM&P data flow in the "A" and "B" message formats. This is required to allow over-fiber operation of an entire network. Work is also required for the 10G Ethernet standard to map any MAC (media access control) messages across this link. Currently the MAC messages are carried out-of-band using 8B/10B violation codes[†], and would have to be mapped into the "A" and "B" messages.

Beyond this, it would be interesting to see if it is possible to construct a distributed sample scrambler version of SDL. This will require some advances in finite field math and the associated circuit implementations in order to allow the receive scrambler to be synchronized without full matrix inversion.

[†] 8B/10B has only a certain number of valid DC balanced words, which form the alphabet of the code space. Violation codes are 10 bit line words which would normally be forbidden, but can be used sparingly (so as not to ruin the DC balance of the code) for signalling.

Appendix A: Examples

A.1 Examples

Some examples of SDL packets are shown in Figure A.1.

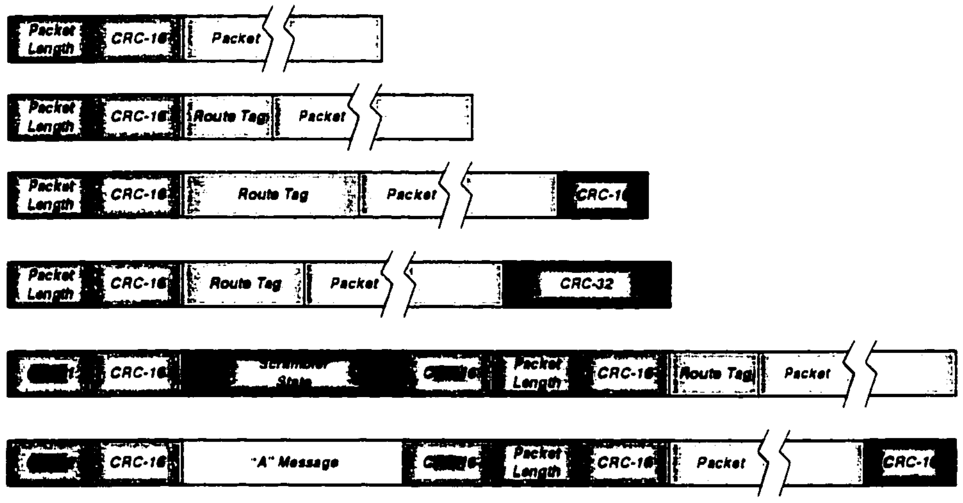


Figure A.1 **SDL packet examples**

Appendix B: Derivation of Ones/Zeroes Density and Transitions Equations

B.1 Ones/Zeroes Density

Given a random binary vector of length n, the probability of having less than m ones in this vector is simply[42]:

$$\frac{\binom{n}{m}}{2^n} \qquad \text{(B.1)}$$

This result is the same for the dual of this problem, namely given a random binary vector of length n, the probability of having less than m zeroes. Thus:

$$p(\#\text{ones} \le m \cup \#\text{zeroes} \le m) = \frac{2\binom{n}{m}}{2^n}$$

or:

$$D(n, m) = 2\binom{n}{m} \qquad \text{(B.2)}$$

where the function $D(n, m)$ returns the number of binary vectors of length n with m or fewer ones or zeroes.

B.2 Transition Density

Given a random binary vector of length n that is repeated ad infinitum, what is desired is an expression for the number of transitions between bits in the vector. For instance, with $n = 7$, the vector 0011110 would have two transitions and three plateaus. This plateau sequence can be represented as 010. Now since this vector is repeated back to back, the number of transitions per n bits would continue to be two. For this discussion, refer to vectors that start and end with the same value as having even form. Consider now vectors with odd form. For example with $n = 7$, the vector 0001111 would also have two transitions per n bits when repeated ad infinitum, but has only two plateaus - 01. Generalizing, one can say that the number of transitions possible in an n bit binary vector repeated ad infinitum is always even.

Now deriving an expression for the number of possible transitions per a repeated binary vector of length n, let the number of transitions be $2t$. Given this, there is an odd vector with $2t$ plateaus and an even vector with $2t + 1$ plateaus that will both generate $2t$ transitions per n bits. Examine the following plateau sequences for an arbitrary vector of length n:

1) 10 - there are $n - 1$ members with this plateau sequence

2) 010 - this can be decomposed into the sum of all plateau sequences of type 10 in length $n - 1$ preceded by a single zero, plus the sum of all plateau sequences of type 10 preceded by two zeroes, etc. constrained by the total length. For instance, with $n = 5$, the following are all the members of the plateau sequence 010:

01000, 01100, 01110

00100, 00110

00010

Generalizing this observation, one can write the number of members of the plateau sequence of length L in a vector of length n as[†]:

$$\#\,\text{members} = \sum_{i=1}^{n-L+1}\left(\sum_{j=1}^{n-L-i+1}\cdots\left(\sum_{k=1} k\right)\right) \qquad L \geq 2 \tag{B.3}$$

Examining this, it is possible to view this as the sum of the volumes of $n - L + 1$ hypercubes of dimension $L - 2$ and size increasing from one to $n - L + 1$. For instance, in the preceding example the hypercubes were of dimension one and there were three of them. Thus one can rewrite eq. (B.3) as:

$$\#\,\text{members} = \sum_{k=1}^{n-L+1} k^{(L-2)} \qquad L \geq 2 \tag{B.4}$$

Now what is desired is an expression for the number of members in the set of

[†] Note that L must be greater than or equal to two or else there will be no transitions in the repeated vector.

vectors of length n that produce $2t$ transitions when repeated ad infinitum. Realizing that a plateau sequence and its bitwise inverse produce the same number of transitions (they are just duals of each other), and that $2t$ transitions can be produced by plateau sequences of length $2t$ and length $2t+1$, one can write:

$$\#\,\text{members} = 2\left(\sum_{k=1}^{n-2t+1} k^{(2t-2)} + \sum_{k=1}^{n-2t} k^{(2t-1)}\right) \qquad t \geq 1$$

or:

$$T(n,t) = 2\left(\sum_{k=1}^{n-2t+1} k^{(2t-2)} + \sum_{k=1}^{n-2t} k^{(2t-1)}\right) \qquad t \geq 1 \qquad \textbf{(B.5)}$$

where the function $T(n,t)$ returns the number of binary vectors of length n that generates t transitions when repeated ad infinitum.

Using James Bernoulli's formula for the sum of a series of constants raised to a fixed power[43][44]:

$$\sum_{k=1}^{n} k^c = \frac{1}{c+1}n^{c+1} + \frac{1}{2}n^c + \frac{c}{2}B_2 n^{c-1} + \frac{c(c-1)(c-2)}{2\cdot3\cdot4}B_4 n^{c-3} + \frac{c(c-1)(c-2)(c-3)(c-4)}{2\cdot3\cdot4\cdot5\cdot6}B_6 n^{c-5} + \ldots \qquad \textbf{(B.6)}$$

where the series terminates on the last positive power of n, and the coefficients B_{2x} are the Bernoulli numbers defined by the generating function:

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!} \tag{B.7}$$

From this equation, a recursion can be derived for the Bernoulli numbers:

$$\sum_{n=0}^{s-1} \binom{s}{i} B_n = 0 \qquad B_0 = 1 \qquad s = 2, 3, 4, \ldots \tag{B.8}$$

which allows any B_n to be calculated[45].

Appendix C: Galois Fields, Linear Feedback Shift Registers, and Cyclic Codes

C.1 Primitive Polynomials and Finite Fields

C.1.1 Galois Fields

A Galois field (or finite field) is a set of symbols which obey a set of restrictions that allow addition, subtraction, multiplication, and division upon them. Here each symbol comes from an "alphabet". For instance, symbols in this alphabet could be nibbles (4 bits), in which case there would be 16 symbols in the alphabet (or they could be bytes, bits, etc.). The reason Galois field mathematics is so important in CRC and error correction coding in digital circuitry is that it allows mathematics to be performed on binary vectors (i.e. bytes) without expanding their size. For instance, adding 2 bytes together must result in another byte, instead of a 9-bit word.

It turns out that the restrictions that we need to impose on the symbols in our set are the same restrictions that define a finite field. These rules were discovered by Evariste Galois, and as a result these fields are called Galois fields.

Basically, a Galois field with "Q" symbols in it ("Q" is called the "order" of the field) is referred to as GF(Q), and has the following simple rules:

1) The elements of the field must form a commutative group under addition. This means that if you take any two elements and add them together, you get another element of the same field (like modulo addition).

Furthermore, they must commute, so $a + b$ must equal $b + a$. Given this, the additive identity element (i.e. you add something to it and get the same thing; similar to 0 in regular math) is labelled "0".

2) The elements of the group without "0" must forr: a commutative group under multiplication. Like in the case of addition, this means that one can multiply any element in the field by any other element, and get another element in the field.

Furthermore, they must also commute, so $a \cdot b$ must equal $b \cdot a$. Given this, the multiplicative identity element (i.e. you multiply something by it and get the same thing; similar to 1 in regular math) is labelled "1".

3) The addition and multiplication operations must distribute. This means that: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

4) The number of elements "Q" in the field must be equal to q^m where "q" is a prime number, and "m" is a positive integer.

Thus if one defines the addition and multiplication operations for ones collection of symbols so that these rules are followed, and the number of elements in the field are equal to q^m, then standard mathematical operations can be used on these symbol elements[†]. This may sound like an easy thing to do, but without some tips, it is very difficult to construct addition and multiplication tables for symbols so that they obey rules (1) and (2).

C.1.2 Addition and Multiplication in a Galois field

Conveniently, it turns out that for Galois fields with a prime number of elements ($m = 1$), one can use modular integer arithmetic to construct the addition and multiplication tables (and as a result, the elements are labelled as integers). However, it must be remembered that these are not real integers, and are only labelled with number symbols for convenience. This becomes especially important when dealing with fields where "Q" is a power of a prime number, as here the elements can no longer be labelled by integer names, and have the mathematical operations mimic the integer math results.

For instance, one can construct the addition and multiplication tables for GF(3) as follows:

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Table C.1 **Addition table for GF(3)**

Similarly for GF(2):

[†] The rules for subtraction and division can be derived from the rules for addition and multiplication. For instance if $a + b = c$, $a = c - b$ etc.

•	0	1	2
0	0	1	0
1	1	1	2
2	0	2	1

Table C.2 **Multiplication table for GF(3)**

+	0	1
0	0	1
1	1	0

Table C.3 **Addition table for GF(2)**

•	0	1
0	0	0
1	0	1

Table C.4 **Multiplication table for GF(2)**

In this particular case, addition becomes the exclusive OR operator, and multiplication becomes the AND operator.

To construct tables for Galois fields of the form GF(q^m), where $m > 1$ it turns out that one can use m-dimensional vectors over the prime order field GF(q). As well, it also turns out that one can also use the roots of certain special polynomials.

In reality, both are required. Polynomial roots are used to help define the multiplication table, and vectors the addition table.

C.1.2.1 Vector Representation of GF(q^m)

As mentioned above, it is possible to use m-dimensional vectors over GF(q) to represent the elements in GF(q^m)[47]†. For example, if one wishes to represent

GF(2^3) = GF(8), it can be represented as a 3D vector of GF(2) elements. For example:

$$GF(8) = \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),\\ (1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$$

In this situation, addition is easy to define; it is performed on an element by element basis according to the rules of GF(q) math. For example $(1,1,0)+(1,0,1) = (0,1,1)$, where the addition table for GF(2) shown in Table C.3 was used for the individual elements. Multiplication however is still not easily defined. For this one has to use polynomial roots.

C.1.2.2 Polynomial Root Representation of GF(q^m)

Polynomial roots provide the alternative method of representing elements of GF(q^m). They also provide a convenient means of performing multiplication, and a way of relating the vector representation to the polynomial root representation. A general polynomial with coefficients from GF(q) is referred to as GF(q)[x] and would be written as:

$$GF(q)[x] = a_0 + a_1 . x + a_2 . x^2 + a_3 . x^3 + \dots$$

where $a_i \in GF(q)$. (Note that the degree of the polynomial can be anything, and has no relation to the elements in GF(q). However, for the vector/polynomial representation of GF(q^m) though, the degree is chosen to be equal to $m - 1$ so that each "tuple" in the vector corresponds to a power of x).

+ GF(q^m) is referred to as an extension field of GF(q).

The basic idea here is that there are two variants of polynomials (both closely related) that can be used to represent the elements of GF(q^m). The first polynomial is of the form:

1) $x^{q^m-1}-1$ (Note that x^n-1 in GF(2)[x] is just x^n+1). It can be shown that successive powers of a primitive root of this polynomial can be used to represent all of the nonzero elements of GF(q^m). In GF(q^m) there are q^m-1 nonzero elements, and one (or more[†]) of these elements are primitive, which means that you can represent every other element in GF(q^m) as a power of this element (with the powers being between 0 and q^m-2).

For instance consider the multiplication table for GF(7):

•	0	1	2	3	4	5	6
0	0	0	0	0		0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Table C.5 **Multiplication table for GF(7)**

[†] Given a field GF(q^m), one can prove that there are always $\phi(q^m-1)$ primitive elements. The function $\phi(t)$ is called the Euler totient function, and is defined as follows: If p_i are all of the divisors of t greater than 1, then $\phi(t) = t\prod_i\left(1-\frac{1}{p}\right)$. This means that there are always primitive elements for any field.

An examination of this table will show that there are two primitive elements: "3" and "5". For instance:

$$5^0 = 1,\quad 5^1 = 5,\quad 5^2 = 4,\quad 5^3 = 6,\quad 5^4 = 2,\quad 5^5 = 3$$

and the final one $5^6 = 1$ brings us back to "1". This property is referred to as the "order" of the element, and in this case "3" and "5" have order six.

2) A primitive m^{th} degree polynomial $p(x) \in GF(q)[x]$. A primitive m^{th} degree polynomial is defined as a polynomial that is irreducible (i.e. not factorable) in $GF(q)[x]$. Furthermore, the smallest degree polynomial of the form $x^n - 1$ that $p(x)$ $m > 1$ is a factor of is $n = q^m - 1$.

For example $x^3 + x + 1$ is primitive in GF(2)[x] as the smallest polynomial of the form $x^n - 1$ that it divides into is $x^7 - 1$, and $7 = 2^3 - 1$.

Now it can be shown that all of the roots of this polynomial have an order of $q^m - 1$[†]. This means that if α is a root of $p(x)$, then one must be able to construct $q^m - 1$ non-zero polynomial representations of powers of the root α using linear combinations of powers of the root α. For instance, if one considers the primitive polynomial given above, and lets α be a root, then $\alpha^3 + \alpha + 1 = 0$ or $\alpha^3 = \alpha + 1$ (remember GF(2)!). Continuing in this fashion, the representations for all of the powers of α can be constructed.

[†] This intuitively makes sense, as the roots of $p(x)$ must be roots of $x^n - 1$, and if any of the roots didn't have order $m - 1$, then $p(x)$ would divide into a smaller polynomial. Note that the roots of $p(x)$ do not lie in the field GF(2), so $x - \alpha$ is not a factor of $p(x)$ in GF(2)[x].

Since these combinations of linear powers of α will include powers from 0 to $m-1$, they can be used to create the vector representation of GF(q^m). For instance, with the primitive polynomial x^3+x+1, one can express all of the powers of α in the form $a\alpha^2+b\alpha+1$, where $a, b, c \in$ GF(2). One can also write a, b, c as the vector (a, b, c) .Thus the addition and multiplication tables for GF(q^m) can now be created.

For example, here is how GF(8) would be constructed:

1) Realizing that $8 = 2^3$, a primitive polynomial over GF(2)[x] is required.

2) Second, it has to be of degree 3, as $m = 3$.

3) Third, one would construct all of the non-zero values of GF(8), by representing them as consecutive powers ($0 \to 6$ of an assumed primitive root of x^3+x+1. Let α be primitive root of x^3+x+1. Thus GF(8) is:

GF(8)
0
α^0
α^1
α^2
α^3
α^4
α^5
α^6

Table C.6 **Exponential Representation of GF(8)**

4) Fourth, one would construct the vector representation of GF(8). Since α is a primitive root of $x^3 + x + 1$, one knows that $\alpha^3 + \alpha + 1 = 0$, or since the coefficients of the polynomial are from GF(2), $\alpha^3 = \alpha + 1$. This gives the required recursion relationship. Thus:

Exponential	**Polynomial**	**Vector** $(1, \alpha, \alpha^2)$
0	0	(0,0,0)
$\alpha^0 = \alpha^7$	1	(1,0,0)
α^1	α	(0,1,0)
α^2	α^2	(0,0,1)
α^3	$\alpha + 1$	(1,1,0)
α^4	$\alpha^2 + \alpha$	(0,1,1)
α^5	$\alpha^2 + \alpha + 1$	(1,1,1)
α^6	$\alpha^2 + 1$	(1,0,1)

Table C.7 **Exponential and vector Representation of GF(8)**

5) Using this table, it is now possible to perform multiplication using addition of powers of exponentials (modulo 7), and addition using vectors (with the individual elements added modulo 2). (It will be seen later that this table can be simply generated by dividing the primitive polynomial into 1 using an LFSR).

C.1.3 Minimal Polynomials and the Factoring of $x^n - 1$

As was mentioned above, the polynomial $x^{q^m - 1} - 1$ and primitive polynomials of degree m from GF(q)[x] are very closely related[48][49]. As a matter of fact it can be shown that if $x^{q^m - 1} - 1$ is factored, the result will be several irreducible polynomials of degree m or smaller. All of these irreducible factors (and product of factors) are called minimal polynomials. Of all the irreducible factors of the polynomials of degree m, some of these will be primitive polynomials. They can only be confirmed through checking that they don't divide into anything less than $x^{q^m - 1} - 1$.

It turns out that by extending the concept of minimal polynomials one can factor $x^n - 1$ where $n \neq q^m - 1$. All that is required is a field where all of the n^{th} roots of unity can be identified. It turns out that fields of the form GF(q^m) that contain the n^{th} roots of unity have the property that n is a factor of $q^m - 1$. For instance GF(27) is the smallest extension field of GF(3) in which one may find the 13th roots of unity. This is required to represent the n^{th} roots of unity as valid polynomials in GF (q^m) [x] in the form $x - \alpha$.

To find these minimal polynomials, there is a trick which makes their construction simple.

1) Let α be a root of a primitive polynomial of degree m in GF(q).

2) Starting with α, find all of the consecutive powers of α^{q^i} where $i = 0 \rightarrow s$ and s is the integer that causes $q^{s+1} \bmod(q^m - 1) = 1$.

3) Next, take the first power of α that wasn't covered by the above (say α^t), and perform the same thing, except this time stop when $tq^{s+1} \bmod(q^m - 1) = t$.

4) Continued this until all of the powers of α have been covered. Each of these sets is called a conjugacy class, and the powers of α form a cyclotomic coset (i.e. $\{\alpha, \alpha^7, \alpha^4, \alpha^{13}\} \leftrightarrow \{1, 4, 7, 13\}$)

5) All of the irreducible (minimal) polynomials have as their roots these powers of α.

For instance, consider the construction of the minimal polynomials for GF(8) shown in Table C.8. These were constructed with the help of Table C.7.

Conjugacy Class	Associated Minimal Polynomial
$\{0\}$	$M_*(x) = (x-0) = x$
$\{\alpha^0 = 1\}$	$M_0(x) = (x-1) = x+1$
$\{\alpha, \alpha^2, \alpha^4\}$	$M_1(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4) = x^3+x+1$
$\{\alpha^3, \alpha^6, \alpha^5\}$	$M_2(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^5) = x^3+x^2+1$

Table C.8 **Minimal polynomials for GF(8)**

The reason for considering this is that this property will be useful in cyclic codes.

C.2 Linear Feedback Shift Registers

Linear feedback shift registers are used to generate scrambler sequences, and are constructed using flip-flops and XOR gates. There are two basic types of LFSR which are duals of each other[50]. The first form is shown in Figure C.1 and

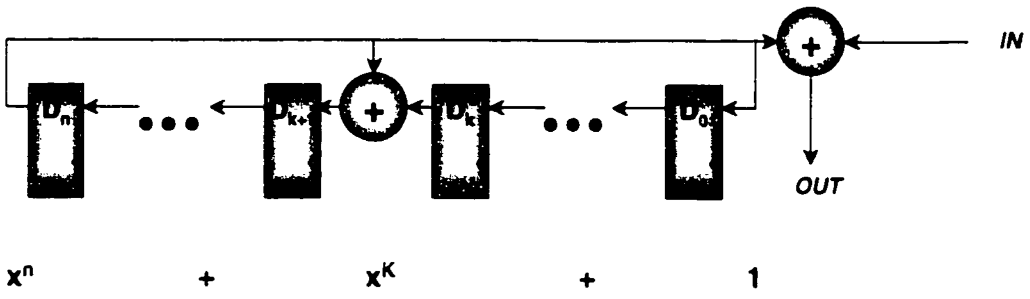


Figure C.1 **LFSR canonical form #1**

corresponds to one stage of long division by a primitive polynomial, and the second form (its dual) is shown in Figure C.2. To see the correspondence between the long

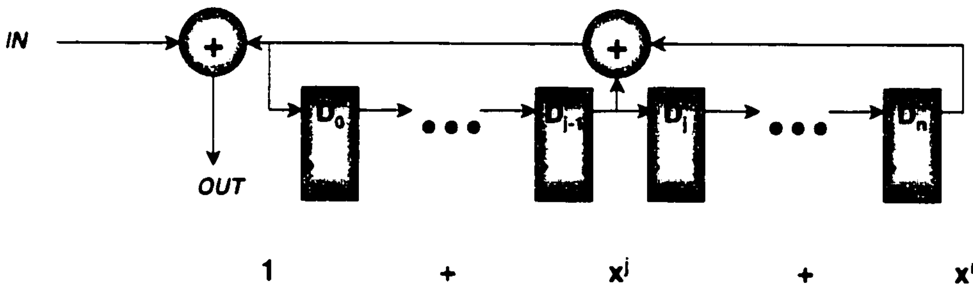


Figure C.2 **LFSR canonical form #2**

division, examine the first step of the example shown in Figure C.3 Here a canonical form #1 LFSR corresponding to the polynomial x^5+x^3+x+1 is shown. As can be seen the first (x^5) and the last tap (1) are always present, and the intermediate terms correspond to the intermediate XOR terms. By inspection, it can be seen that the contents of the LFSR correspond to the remainder of the division after each stage[†]. The presence of a "1" in the leftmost flop indicates whether the polynomial should be subtracted or not, and the last term always gets the LSB term of the

[†] Remember that in GF (2) addition and subtraction are the same: XOR.

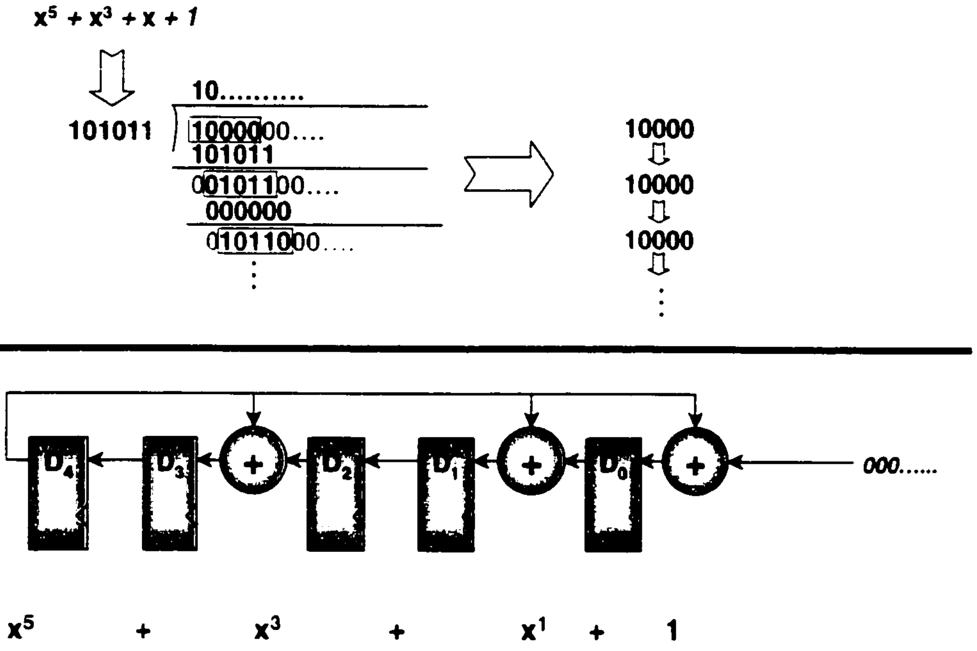


Figure C.3 **Binary division example**

polynomial as the remainder shifts to the right.

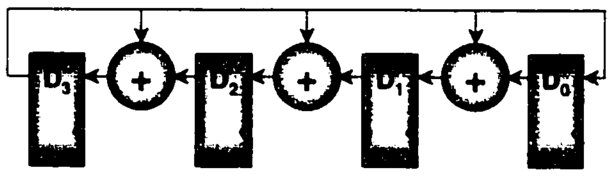
In this figure, the LFSR is shown as having an input stream of consecutive zeroes, which is the same as having no input. In reality though, with an input, this circuit would implement finite field division, and as a result this circuit is used for CRC calculations. With a zero input, it performs repeated division upon its contents.

The result is that the output of the LFSR is a maximal length sequence of length $2^m - 1$. It can in turn be shown that each flop in either of these 2 canonical forms generates the same maximal length sequence, just at different offsets[29].

C.2.1 Parallel LFSR Operations

Up until this point, we have only shown how to implement an LFSR based scrambler in bit-serial fashion. Now we will consider how to implement this operation in parallel.

Consider the form #1 implementation of an LFSR based on the polynomial $x^4 + x^3 + x^2 + x + 1$:



x⁴ + x³ + x² + x¹ + 1

Figure C.4 **Canonical form #1 LFSR example**

To speed up this process, this structure can be "parallelized" by writing the state transition table for this machine. Letting the leftmost D-flop output be D_3, and the rightmost be D_0, the transition equations for this machine become:

$$\begin{aligned} D_3^+ &= -D_3 + D_2 \\ D_2^+ &= -D_3 + D_1 \\ D_1^+ &= -D_3 + D_0 \\ D_0^+ &= -D_3 \end{aligned}$$

Ignoring the negative signs (as they do nothing in GF(2)), these equations can be written in matrix form as:

$$\begin{bmatrix} D_3^+ \\ D_2^+ \\ D_1^+ \\ D_0^+ \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} D_3 \\ D_2 \\ D_1 \\ D_0 \end{bmatrix}$$

This can be written in vector form as:

$$\vec{D}_{n+1} = \vec{T} \cdot \vec{D}_n$$

Thus one can write the state of the machine at 4 cycles into the future as:

$$\begin{aligned} \vec{D}_{n+4} &= \vec{T} \cdot \vec{D}_{n+3} \\ &= \vec{T} \cdot (\vec{T} \cdot \vec{D}_{n+2}) \\ &= \vec{T}^4 \vec{D}_n \end{aligned} \tag{C.1}$$

This would implement a parallel LFSR that would update 4 bits at a time.

The "magic" with this is that given the transition matrix $\vec{T}$ for the LFSR one can explicitly write the transformation forwards and backwards in time as $\vec{T}^{i}$. In circuit form, this transformation appears to be a collection of XOR terms (for an n bit LFSR, the average number of XOR terms per bit will be $n/2$) with the current state as their input.

C.2.2 Vector Representation of GF(q^m)

It actually turns out that the sequence that a form #1 LFSR produces when starting at 1 is actually the vector representation of the Galois field. For instance, the sequence from the example in Table C.7 using the primitive polynomial $x^3 + x + 1$, could be generated by the following machine (where the machine would be initialized so that the rightmost D-flop contained 1):

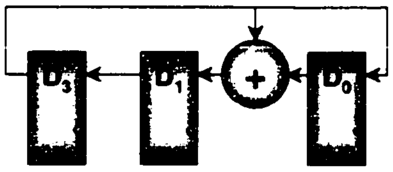


Figure C.5 **LFSR generator for vector representation of GF(8)**

The only difference between this output, and sequence in Table C.7, is that this sequence is backwards, as this machine progresses from right to left, as opposed to left to right. Here the output would be the values of the D-flops after each shift.

C.2.3 Matrix Representations of the Canonical LFSR Forms

Given this information, one can represent the two canonical forms as transition matrices. For form #1, with a primitive polynomial of the form:

$$x^{n+1} + a_n x^n + \dots + a_2 x^2 + a_1 x + 1$$

and the flops labelled as in Figure C.1, the following transition matrix would apply:

$$\hat{T} = \begin{bmatrix} a_n & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 & 0 & \dots & 0 & 0 \\ a_1 & 0 & \dots & 1 & 0 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix} \qquad \text{(C.2)}$$

Similarly, for form #2 we have:

$$\hat{T} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & a_1 & a_2 & \dots & a_n \end{bmatrix} \qquad \text{(C.3)}$$

As can be seen, they are duals of each other (substitute $a_{(n-i)(n-j)}$ for a_{ij}). A final point of interest is that since the sequence repeats after $2^m - 1$ shifts, one can write:

$$\hat{T}^{(2^m - 1)} = I$$
$$\hat{T}^{2^m}\hat{T}^{-1} = I$$
$$\hat{T}^{2^m} = \hat{T}$$

C.3 Linear Block Codes

A block code is a code that takes k message symbols and encodes them into n code symbols. This would be described as an (n, k) block code. Here each symbol comes from an "alphabet". For instance, symbols in this alphabet could be nibbles (4 bits), in which case there would be 16 symbols in the alphabet (or they could be bytes, bits, etc.). Thus if there were Q symbols in the alphabet, there would be q^k codewords, out of a total of Q^n possibilities. These Q^n possibilities can be viewed as an n-dimensional vector space, and the Q^k codewords can be viewed as coordinates in this vector space.

In proper mathematical language, this would be stated as the Q^n possibilities form a vector space over the set of symbols.

Now the key to making this a good code is spreading these codewords out as far as possible from each other, so that when data corruption occurs (say during transmission), it will be as difficult as possible for this corruption to cause the received word to look like another codeword. The idea behind this is that a code is designed so that it is capable of detecting a certain number of errors, and correcting a lesser amount of these errors. This can be viewed as looking at the received codeword, and deciding which actual codeword it is closest to. If the errors introduced during transmission cause this received codeword to be closer to another valid codeword, then a decoding error will occur, but at least the errors will be detectable. The minimum number of codewords that have to change to make one codeword look like another is referred to as the "minimum distance" of the code. Thus a code with a minimum distance $d_{\min}$, can detect $d_{\min} - 1$ errors and correct $(d_{\min} - 1)/2$ errors.

A block code linear is if the collection of q^k codewords (remember the code is of the form (n, k), where k message symbols map into "n" codewords) form a vector subspace over the n-dimensional space made up of GF(Q) n-tuples. This means that any linear combination of codewords will always be within the n-dimensional space, and the codeword will be formed from linear combinations of the message word. (Note that we have not placed a requirement on the space of codewords to be a finite field, as the number of symbols (Q) is not necessarily a prime number. We are merely treating this as a finite vector space.)

For example, if we have a k-dimensional message vector $\vec{m}$, then the n-dimensional codeword $\vec{c}$ can be written as:

$$\vec{c} = \vec{m}\vec{G}$$

where $\vec{G}$ is $k \times n$ matrix called the generator matrix. For example:

$$\vec{G} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{0,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{n-1,n-1} \end{bmatrix}$$

Thus $c_0 = m_0 g_{0,0} + m_1 g_{1,0} + \ldots + m_{k-1} g_{k-1,0}$, etc. Here all of the addition and multiplication is performed in GF(Q).

C.3.1 Cyclic Codes

A cyclic code is a special type of linear block code. The difference is that cyclic codes are not just treated as vectors in an n-dimensional space made up of GF(Q) n-tuples. Instead the code words are forced to obey the following rule:

1) An (n, k) linear block code is said to be cyclic if every left and right barrel shift of a codeword is also a codeword. For example, if $\vec{c} = (c_0, c_1, \ldots, c_{n-1})$, then $\vec{c}' = (c_{n-1}, c_0, c_1, \ldots, c_{n-2})$ is also a codeword.

This means that we can now represent the codeword $\vec{c} = (c_0, c_1, \ldots, c_{n-1})$ as a polynomial:

$$c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$$

and we are then able to express the cyclic property as:

$$c'(x) = x \cdot c(x) mod(x^n - 1)$$

For instance, if $c(x) = c_0 + c_1 x + c_2 x^2$, then $x \cdot c(x) mod(x^3 - 1)$ would be the remainder of:

$$Rem(x^3 - 1 \sqrt{c_2 x^3 + c_1 x^2 + c_0 x}) = c_1 x^2 + c_0 x^2 + c_2$$

which is correct.

The point to all of this is that when one expresses cyclic code words as being modulo $(x^n - 1)$, it is possible to treat the entire code word as something pretty close to a Galois field. This "something pretty close" is called a "ring of polynomials" GF(Q)[x] modulo $(x^n - 1)$ and is written GF(Q)[x]/$(x^n - 1)$, and is a commutative ring with identity.

To illustrate this difference, consider integer math modulo q. If q is a prime number, the result is a Galois field. If q is not prime, the result is a commutative ring with identity[†]. As a result, GF(Q)[x]/ $f(x)$ would be a field if $f(x)$ was irreducible (i.e. it couldn't be factored in GF (Q)[x].

The reason for wanting to treat the codewords as a ring of polynomials, is that it is

[†] The formal difference is that all of the elements in a field (except "0") must have a multiplicative inverse.

possible to bring a lot of algebra to bear on the problem of designing good codes.

First redefine cyclic codes as:

1) C is a Q-ary cyclic code of length n if and only if the polynomials in C form an ideal in GF(Q)[x]/(x^n - 1).

An "ideal" I, is a subset of elements in a ring R, that has the magic property that there exists some generator element g, (also a member of I) such that the product of any element of the ring R multiplied by g produces an element in I. This is illustrated below:

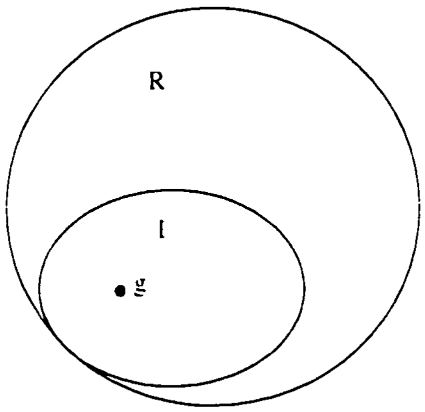


Figure C.6 **Illustration of ideal concept**

Given this definition of cyclic codes, one can now state the following:

1) The generator polynomial g is a monic minimal polynomial from I (monic means a leading coefficient of 1; i.e. x^n would be the first coefficient), and has a degree r less than n (the code's length - which when viewed as a polynomial is the peak degree of a codeword in I).

2) The generator polynomial is a factor of $x^n - 1$.

3) Each code polynomial $c(x)$ can be expressed as $c(x) = m(x)g(x)$ where $g(x)$ is the generator polynomial and $m(x)$ is the message polynomial of degree k less than $n - r$.

For example, consider construction of a binary (GF(2)) cyclic code of length 7. To do this a generator polynomial that divides $x^7 - 1$ is needed. Conveniently, the minimal polynomials are given in Table C.8, so any combination of these polynomials as the generator polynomial can be chosen. Say $g(x) = (x^3 + x + 1)(x + 1) = x^4 + x^3 + x^2 + 1$ is chosen as the generator polynomial. Thus the degree of the message polynomials must be less than or equal to two. As a result the polynomials in Table C.7 can be used as the representation for the message polynomials. Thus one can construct the following set of codewords:

The final step in cyclic coding, is to make the code systematic. This means that in an (n, k) code, the k bits consist of the untranslated message, and the remaining r bits are the check bits. Letting the cyclic code be constructed so that the first r bits are the check bits, and the remaining k bits are the message bits, the polynomial corresponding to the message bits will be $x^{n-k}m(x) = (0, \ldots, 0, m_0, \ldots, m_{k-1})$. Dividing this by $g(x)$, one obtains:

$$x^{n-k}m(x) = q(x)g(x) + m(x) \bmod g(x)$$

$m(x)g(x)$	**Code polynomial**	**Code Word**
$0 \cdot g(x)$	0	(0000000)
$1 \cdot g(x)$	$1 + x^2 + x^3 + x^4$	(1011100)
$x \cdot g(x)$	$x + x^3 + x^4 + x^5$	(0101110)
$x^2 \cdot g(x)$	$x^2 + x^4 + x^5 + x^6$	(0010111)
$(x^2 + 1) \cdot g(x)$	$1 + x^3 + x^5 + x^6$	(1001011)
$(x^2 + x + 1) \cdot g(x)$	$1 + x + x^4 + x^6$	(1100101)
$(x + 1) \cdot g(x)$	$1 + x + x^2 + x^5$	(1110010)
$(x^2 + 1) \cdot g(x)$	$x + x^2 + x^3 + x^6$	(0111001)

Table C.9 **Binary (7,3) cyclic code words**

Subtracting the remainder $r(x) = m(x) \mathrm{mod}\, g(x)$ from $x^{n-k}m(x)$ we have:

$$q(x)g(x) = x^{n-k}m(x) - r(x) = (-r_0, \ldots, -r_{n-k-1}, m_0, \ldots, m_{k-1}) \qquad \text{(C.4)}$$

Note that since this was formed by multiplying $q(x)$ by $g(x)$, this must be a valid code word. This gives the desired result: namely that one can send $(-r_0, \ldots, -r_{n-k-1}, m_0, \ldots, m_{k-1})$, and decode it as a normal code word, reading the message directly.

Thus the algorithm for encoding a systematic cyclic code is just simply dividing the shifted message polynomial and inserting the remainder as the check sequence. It turns out that this division can be accomplished using a linear feedback shift register, such as those shown in Figures C.1 - C.2.

Appendix D: How To Sabotage a POS Link

D.1 The POS Standard

Packet over SONET/SDH as specified in RFC 1619[16] utilizes byte escaping as per RFC1662[17] followed by an $x^{43} + 1$ self-synchronous scrambler. This payload is mapped into the SONET/SDH payload in any of the standard formats, up to the entire payload size (i.e. OC-48 would allow up to STS-48c: 2488 Mb/s). Finally SONET/SDH imposes an $x^7 + x^6 + 1$ frame synchronized scrambler on top of the SONET frame[7].

Now since SONET/SDH is a TDM (time division multiplex) interleaved, there is no real chance of an attack unless the entire SONET/SDH payload is devoted to carrying the POS link. However this is precisely the mode of operation that many of the network operators are considering using POS links in. This is the "big, fat pipe" approach, and is driven from the desire to handle all of the data switching and OAM&P (operations, administration, maintenance, & provisioning) in the network.

Consider the network diagram shown in Figure D.1. Here routers are connected as a logical mesh network using add-drop multiplexers, with the SONET/SDH channelization providing the "mesh". The issue with this network is that the channelization scheme is semi-static[†], and as a result, dynamic reconfiguration of

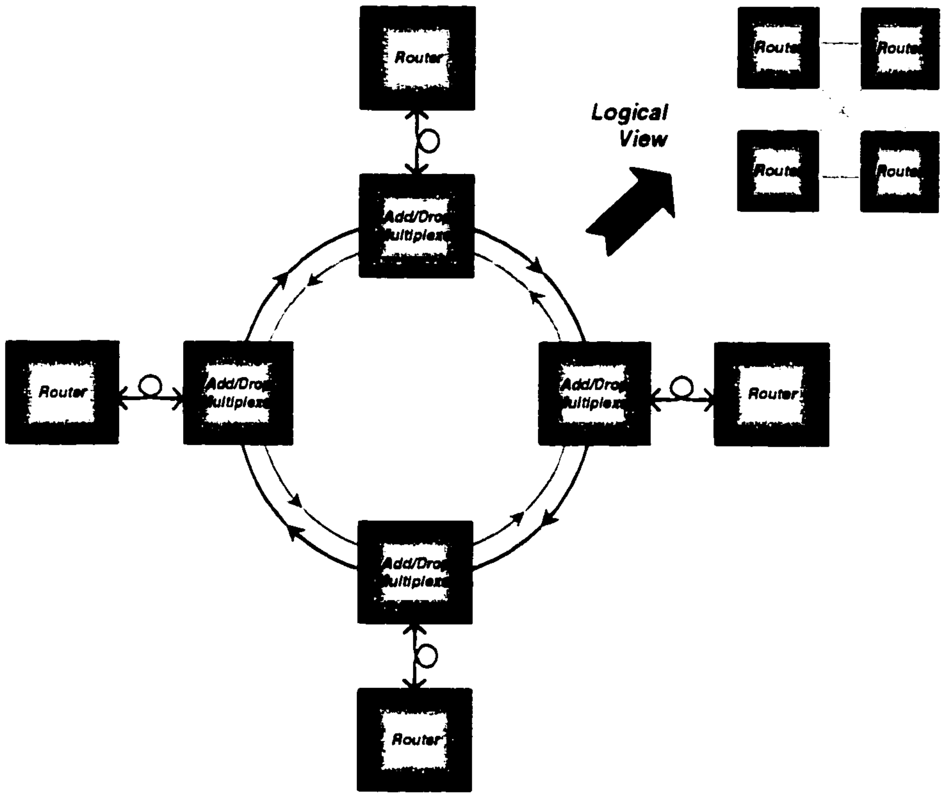


Figure D.1 **Ring based router network**

the network is not typically done. The second problem is that the OAM&P schemes for the two types of networks are totally incompatible and are typically implemented separately. The problem with this is that all of this equipment comprises one network, and is maintained by one network operator. Consequently the "dual OAM&P" paths are very unappealing, as it makes fault isolation difficult in the event of a failure.

To solve both of these issues, the network shown in Figure D.2 is currently being

† SONET/SDH was designed for circuit switched networks transporting voice circuits. The entire network is essentially configured as a patch panel, and in order to reconfigure it, a manual protection switch is implemented, and then the working channel reconfigured, followed by a revert.

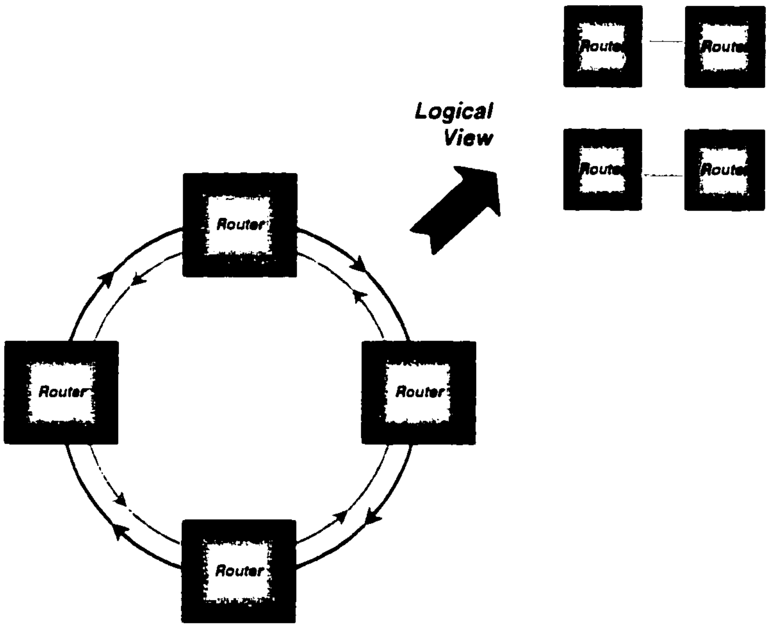


Figure D.2 **"Big, fat pipe" network**

marketed as the solution. It consists of using the SONET/SDH data links only for point-to-point connections, and having the router fabric handle all of the pass-through traffic. The problem with this type of network, which is the network that POS is being positioned for, is that it uses the complete SONET/SDH payload for data transport. As well it is a point-to-point approach, which means that the path (the payload in a SONET/SDH frame) is always aligned in the fourth column (see Section D.2.1). This leaves the door open for an attack on POS.

D.2 Attacking the $x^{43} + 1$ Scrambler

The key to attacking the $x^{43} + 1$ scrambler is to create packets that consist of a random 43 bit vector, followed by a long string of zeroes. This will have the effect of randomly initializing the scrambler state, and the string of zeroes will cause this state to be repeated for the duration of the zeroes. Using eqs. (4.2) and eqs. (4.3), we can calculate on average how many packets we will have to send in order to arrive at a packet that produces a certain DC balance skew or a certain low number of transitions. Finally, these packets will have to have the SONET/SDH frame scrambler sequence added to them (modulo 2). This sequence is given below and was generated using the primitive polynomial $x^7 + x^6 + 1$ as specified in GR-253[7]:

```
FE 04 18 51 E4 59 D4 FA 1C 49 B5 BD 8D 2E E6 55
FC 08 30 A3 C8 B3 A9 F4 38 93 6B 7B 1A 5D CC AB
F8 10 61 47 91 67 53 E8 71 26 D6 F6 34 BB 99 57
F0 20 C2 8F 22 CE A7 D0 E2 4D AD EC 69 77 32 AF
E0 41 85 1E 45 9D 4F A1 C4 9B 5B D8 D2 EE 65 5F
C0 83 0A 3C 8B 3A 9F 43 89 36 B7 B1 A5 DC CA BF
81 06 14 79 16 75 3E 87 12 6D 6F 63 4B B9 95 7F
02 0C 28 F2 2C EA 7D 0E 24 DA DE C6 97 73 2A
```

In the SONET/SDH frame structure, this scrambler is reset on each frame and begins on the first byte of row 4, column 4.

D.2.1 The SONET/SDH Frame Structure

SONET and SDH are essentially identical[†], and can be viewed as stacked STS-1 (51 Mb/s) frames as shown in Figure D.3. Here an STS-3c (155 Mb/s) frame is shown composed of 3 STS-1 frames. As is indicated, the byte order of transmission is front to back on a row by row basis, (MSB to LSB). For higher order frames, like

[†] The SDH frame structure as defined by the ITU was based on GR-253, and for this discussion is essentially identical, aside from naming conventions. As a result the SONET version of the frame structure will be used.

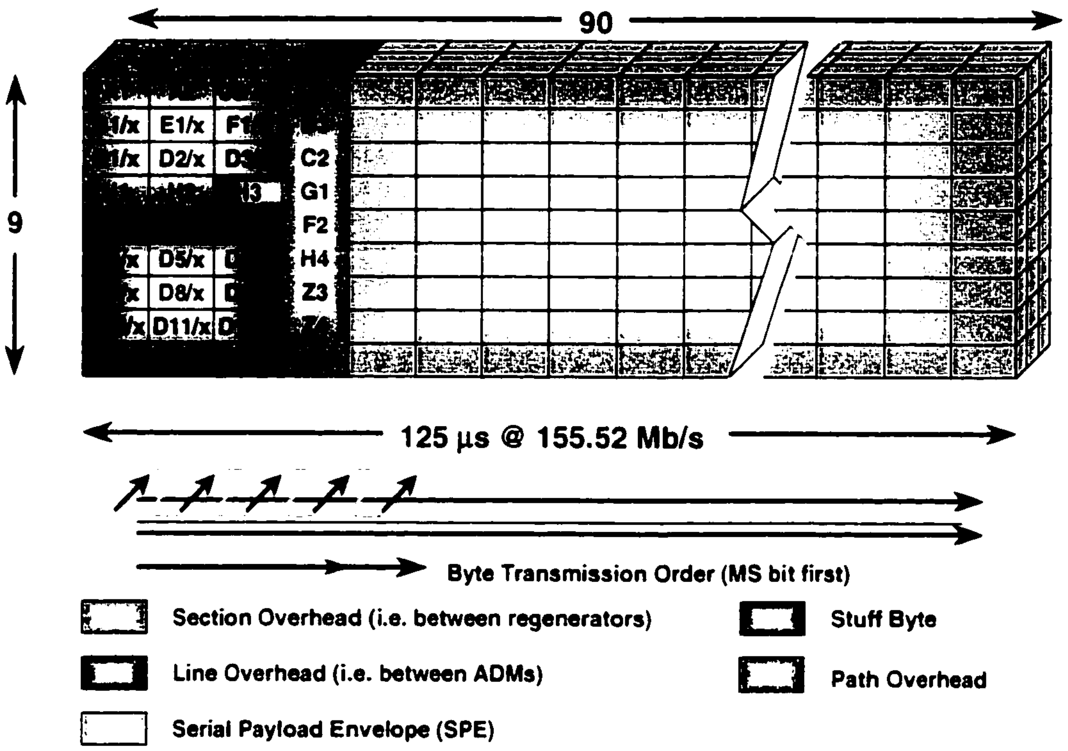

90
9
E1/x
D2/x
C2
G1
F2
H4
Z3
D5/x
D8/x
D11/x
125 µs @ 155.52 Mb/s
Byte Transmission Order (MS bit first)
Section Overhead (i.e. between regenerators)
Stuff Byte
Line Overhead (i.e. between ADMs)
Path Overhead
Serial Payload Envelope (SPE)

Figure D.3 **STS-3 frame structure viewed as 3 stacked STS-1 frames at point of generation**

STS-12c (622 Mb/s) and STS-48c (2488 Mb/s), the structure is similar, with this one difference†: in concatenated payloads higher than STS-3c, data is mapped into the last 2/3 of the path overhead bytes (column 4). What this means is that the first $N/3$ rows in an STS-*N*c contain 86 bytes of data, and the remaining rows contain 87 bytes of data. Thus we have the following:

Frame	Payload bytes/frame
STS-3c	2322
STS-12c	9360
STS-48c	37440

Table D.1 **Number of payload bytes per SONET frame type**

† Actually there is another difference, and this has to do with the line order of transmission for the STS-1 frames that make up the higher order frame, but this is not germane to this discussion.

Thus the probability of a random packet aligning correctly to a SONET/SDH frame is the inverse of the number of bytes per frame. If a packet (as described above) does this, the effect will be to cancel out the SONET/SDH frame scrambling for the duration of the packet.

D.2.2 Attack Probabilities

Given these probabilities, the number of packets required on average to produce the following DC balance offsets and minimum transitions are given in Tables D.2 - D.7

Maximum Number of Bits/Scrambler Period	Average Number of Packets to Achieve
0	3.31739e+015
1	1.60877e+014
2	7.47476e+012
3	5.32661e+011
4	5.17828e+010
5	6.43922e+009
6	9.8372e+008

Table D.2 **DC Balance attack probabilities for STS-3c payload**

Maximum Number of Transitions/Scrambler Period	Average Number of Packets to Achieve
2	3.31739e+015
4	7.84014e+012
6	1.12102e+010
8	1.47283e+007

Table D.3 **Transition attack probabilities for STS-3c payload**

for STS-3c to STS-48c payloads. As can be seen, the number of packets to achieve

Maximum Number of Bits/Scrambler Period	Average Number of Packets to Achieve
0	5.1225e+015
1	5.87476e+014
2	3.0277e+013
3	2.14808e+012
4	2.08737e+011
5	2.59564e+010
6	3.96538e+009

Table D.4 **DC Balance attack probabilities for STS-12c payload**

Maximum Number of Transitions/Scrambler Period	Average Number of Packets to Achieve
2	5.1225e+015
4	3.1564e+013
6	4.5188e+010
8	5.93698e+007

Table D.5 **Transition attack probabilities for STS-12c payload**

Maximum Number of Bits/Scrambler Period	Average Number of Packets to Achieve
0	1.14136e+017
1	4.43784e+015
2	1.20523e+014
3	8.57757e+012
4	8.34947e+011
5	1.03828e+011
6	1.58615e+010

Table D.6 **DC Balance attack probabilities for STS-48c payload**

the desired outcome is very high. For instance, the average length of time to achieve 8 or fewer transitions in an STS-3c link for one frame (2322 bytes) on a T1 (1.544

Maximum Number of Transitions/Scrambler Period	Average Number of Packets to Achieve
2	1.14136e+017
4	1.26256e+014
6	1.80759e+011
8	2.37479e+008

Table D.7 **Transition attack probabilities for STS-48c payload**

Mb/s) connection exclusively dedicated to this pursuit would be roughly 49.22 hours.

While this long time may be perceived as providing protection against this form of attack, the increasing prevalence of higher speed connections, and many hackers operating in tandem will significantly decrease these times.

D.2.3 Line Spectra

The other effect of achieving frame synchronization with a long packet as described above is to effectively reduce the power spectral density as seen by the timing recovery circuit to a line spectra with a null at the $f_s/2$: the area of the spectrum where all of the timing information lives. While the power of the individual spectral lines will vary from scrambler state to scrambler state, the average power will be as seen in Figure D.4.

D.3 Attack Efficacy

The natural question to ask at the end of this discussion is: "OK - so I achieve one of these conditions on a SONET/SDH link - so what?". The answer is that it is

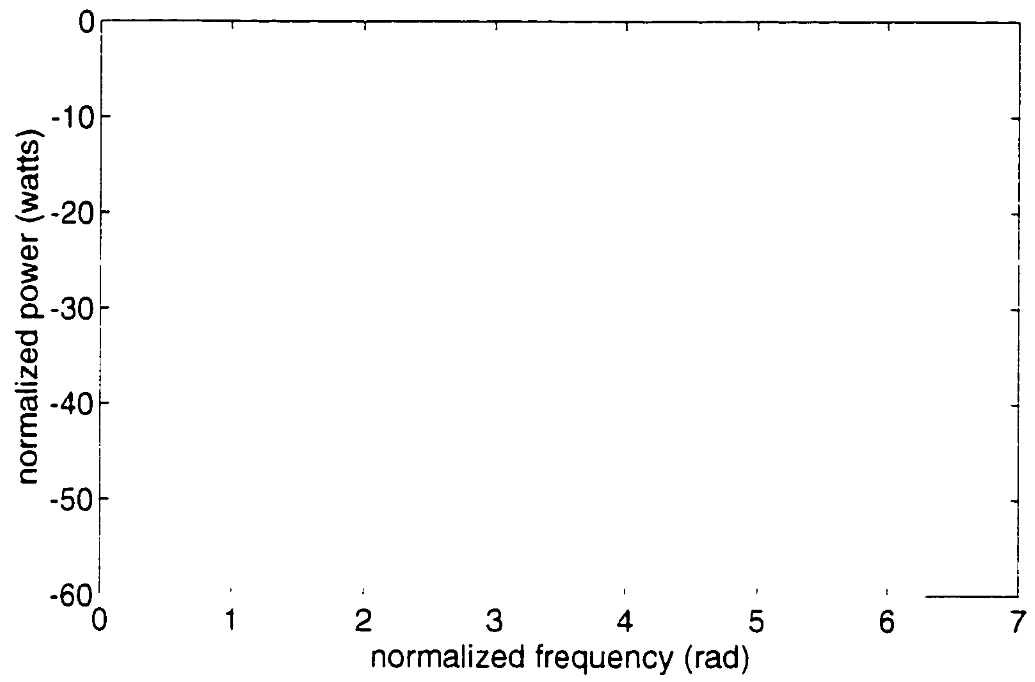


Figure D.4 **Average power spectral density of scrambler output during a long run of zeroes**

totally dependent upon how the timing recovery and front-end of the SONET/SDH equipment was implemented. The specifications for SONET/SDH link timing recovery given in G.958[46] merely specify that the timing recovery circuit must be capable of receiving 72 consecutive zeroes and still meet the jitter accommodation requirements (0.1 UI)[†]. Anything beyond this specification is equipment specific. As a result, each system must be independently analyzed.

[†] SONET/SDH line systems are subject to three jitter specifications: jitter accommodation which refers to the ability of the receiver to deal with jittered input, jitter generation which refers to the output jitter, and jitter transfer which applies to repeaters. In all cases, jitter is specified in terms of UI or unit intervals of the bit period.

Appendix E: Explanation of Header CRC Transform

E.1 Discussion

The header CRC-16 calculation involves dividing a polynomial in GF(2) consisting of the 16 bit length field multiplied by x^{16} times a generator polynomial $g(x) = x^{16} + x^{12} + x^{5} + 1$. If the length field contains 16 bits labelled $L_{15}:L_0$, this can be expressed as:

$$\mathrm{CRC16} = \frac{L_{15}.x^{31} + L_{14}.x^{30} + \ldots + L_1.x^{17} + L_0.x^{16}}{x^{16} + x^{12} + x^{5} + 1} \tag{E.1}$$

As discussed in Section C.2, it can be observed that this division operation, when performed using classic long division can be automated using a linear feedback shift register (LFSR) whose taps correspond to the non-zero elements of the generator polynomial $g(x)$. For $g(x) = x^{16} + x^{12} + x^{5} + 1$ this is shown in Figure E.1 In

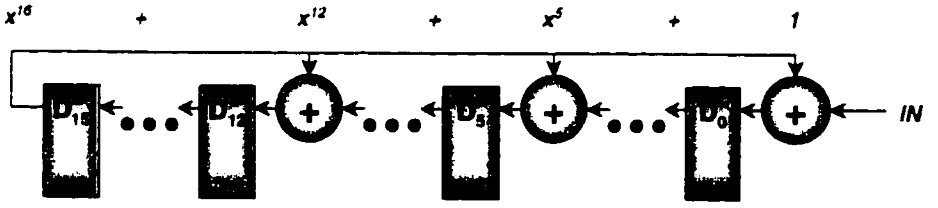


Figure E.1 **LFSR implementation of CRC16 division operation**

normal operation the polynomial to be divided would be shifted in MSB first, and when the last bit was shifted in, the remainder would be contained in $\vec{D}$. As per Section C.2.1, it can be seen that the LFSR update can be written as a transition matrix where the state of the LFSR is $\vec{D}$, the transition matrix is $\vec{T}$, and the input is $\vec{X}$. Thus:

$$\vec{D}^{+} = \vec{T} \cdot \vec{D} + \vec{X} \tag{E.2}$$

and:

$$\vec{T} = \left[\begin{array}{cccccccccccccccc}
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 \\
1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & 1 \\
- & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & 1 \\
- & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1
\end{array}\right] \tag{E.3}$$

where the first row corresponds to D_{0}.

Taking this 16 bits at a time and realizing that the length of the input is only 32 bits, we can write[÷]:

$$\mathrm{CRC} = \vec{T}^{16} \cdot (\mathrm{Length}) \qquad \text{(E.4)}$$

Since math in the field of GF(2) is linear, this implies a unique CRC for every unique 16 bit length. Consequently, the probability of hitting a false 32 bit sequence in random data that corresponds to a valid SDL header is 2^{-16}.

[†] The way to see this is to realize that the first 16 shifts of input data do nothing except load the LFSR with the data, and the next sixteen shifts implement the $\vec{T}^{16}$ operation. Since the lower 16 bits of the input sequence are zero, we are just left with the $\vec{T}^{16}$ operation.

Appendix F: SDL Specification

F.1 Specifications

F.1.1 Formats

Requirement <#1>: General Frame Format

The format of an SDL frame consists of a 16 bit length field, followed by a 16 bit CRC-16 check field. This allows frame boundary detection, and framing. This is shown in Figure F.1.

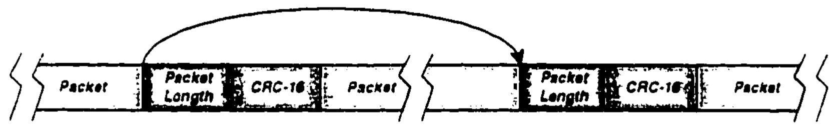


Figure F.1 **SDL frame boundary delineation**

Requirement <#2>: Bit Order of Transmission

The bit order of transmission is MSB to LSB on a per byte basis, with the MS byte being transmitted first.

Requirement <#3>: Byte Alignment

The SDL link shall support byte boundary alignment, and not bit boundary alignment. Consequently the number of bits in the packet must be a multiple of 8.

Requirement <#4>: Length Field

The length field is the 1st and 2nd bytes of the SDL header. The 16 bit length value is placed with the least significant bit right justified in the length field.

Requirement <#5>: Maximum Packet Size

The maximum packet size allowed is 65336 bytes. This corresponds to a length field value of 0xFFFF.

Requirement <#6>: Minimum Packet Size

The minimum packet size is 4 bytes. This corresponds to a length field value of 0x0004.

Requirement <#7>: Header CRC Field

The header CRC-16 field is the 3rd and 4th bytes of the SDL header, as shown in Figure F.2.

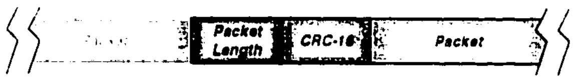


Figure F.2 **SDL header**

The CRC-16 is used to detect multi-bit errors and correct single bit errors in the SDL header. The CRC-16 field is filled with the value of a CRC calculation which is performed over the first two bytes of the header. The CRC-16 field shall contain:

1) the remainder of the division (modulo 2) by the generator polynomial of the product of x^{16} by the 2 length field bytes over which the CRC is calculated.

The CRC-16 generator polynomial is:

$$g(x) = x^{16} + x^{12} + x^{5} + 1$$

The result of the CRC calculation is placed with the least significant bit right justified in the CRC field.

Requirement <#8>: DC Balance

In order to maintain DC balance on the SDL link, the length field and the header CRC field shall be XORed with the 32 bit value 0xB6AB31E0[†]. This value is the maximum transition, minimum autocorrelation sidelobe, DC-balanced sequence of length 32.

Requirement <#9>: Route Tag

The SDL link shall support a provisionable offset to the length field to allow for the attachment of layer 2 routing information (i.e. MPLS tags).

Optional Requirement <#10>:Supported Route Tag Values

The recommended range of provisionable values is given in Table F.1.

Recommended Offset Number	Byte Offset
0x0	0
0x1	1
0x2	2
0x3	3
0x4	4
0x5	5
0x6	6
0x7	7

Table F.1 **Length offset values**

[†] This is the maximum transition, minimum sidelobe, DC balanced Barker-like sequence of length 32.

Recommended Offset Number	Byte Offset
0x8	8
0x9	10
0xA	12
0xB	14
0xC	16
0xD	20
0xE	24
0xF	32

Table F.1 **Length offset values**

Requirement <#11>:Payload

The packet data and route tag data (both indistinguishable to the SDL engine) shall be inserted on a byte-by-byte basis after the header CRC-16 in the order in which it is received.

Requirement <#12>:Payload CRC

The SDL link shall support the provisionable insertion of a CRC-16 or CRC-32 calculated over the packet data and route tag data (both indistinguishable to the SDL engine). If provisioned, these 2 or 4 byte payload CRC fields shall be placed after the last byte of packet data.

Requirement <#13>:Payload CRC-16

The optional payload CRC-16 is used to detect bit errors within the payload. The 2 byte CRC field is filled with the value of a CRC calculation which is performed over the entire contents of the payload (route tag + packet). The CRC field shall contain the ones complement sum (modulo 2) of:

1) the remainder of $x^k(x^{15} + x^{14} + \ldots + x + 1)$ divided (modulo 2) by the generator polynomial, where k is the number of bits of the information over which the CRC is calculated; and

2) the remainder of the division (modulo 2) by the generator polynomial of the product of x^{16} by the information over which the CRC is calculated.

The CRC-16 generator polynomial is:

$$g(x) = x^{16} + x^{12} + x^5 + 1$$

The result of the CRC calculation is placed with the least significant bit right justified in the CRC field.

Requirement <#14>:Payload CRC-32

The optional payload CRC-32 is used to detect bit errors within the payload. The 4 byte CRC field is filled with the value of a CRC calculation which is performed over the entire contents of the payload (route tag + packet). The CRC field shall contain the ones complement sum (modulo 2) of:

1) the remainder of $x^k(x^{31} + x^{30} + \ldots + x + 1)$ divided (modulo 2) by the generator polynomial, where k is the number of bits of the information over which the CRC is calculated; and

2) the remainder of the division (modulo 2) by the generator polynomial of the product of x^{32} by the information over which the CRC is calculated.

The CRC-32 generator polynomial is:

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The result of the CRC calculation is placed with the least significant bit right justified in the CRC field.

F.1.2 Special Payloads

Requirement <#15>:Interpacket Fill

A length field value of 0x0000 shall be used to designate interpacket fill. This is shown in Figure F.3.

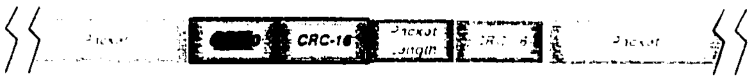


Figure F.3 **Fill header**

Requirement <#16>:Special Payloads

A length field value of 0x0001, 0x0002, and 0x0003 shall correspond to a special 8 byte packet consisting of a 6 byte payload and a CRC-16 calculated over the 6 bytes of the special payload. This is shown in Figure F.4. These special payloads are used

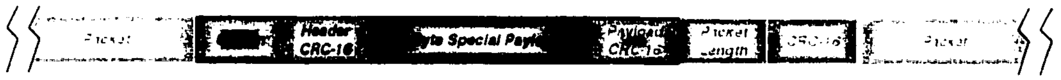


Figure F.4 **Special payload packet format**

for scrambler state transmission and OAM message transmission. For framing purposes, these packets have a length of 0x0008.

Requirement <#17>:Special Payload CRC-16

The special payload CRC-16 is used to correct single bit errors and detect multi-bit errors within the 6 byte special payload. The 2 byte special payload CRC field is filled with the value of a CRC calculation which is performed over the 6 bytes of the payload. The CRC field shall contain the ones complement sum (modulo 2) of:

1) the remainder of $x^k(x^{15} + x^{14} + \ldots + x + 1)$ divided (modulo 2) by the generator polynomial, where k is the number of bits of the information over which the CRC is calculated; and

2) the remainder of the division (modulo 2) by the generator polynomial of the product of x^{16} by the information over which the CRC is calculated.

The CRC-16 generator polynomial is:

$$g(x) = x^{16} + x^{12} + x^5 + 1$$

The result of the CRC calculation is placed with the least significant bit right justified in the CRC field.

F.1.3 Scrambling

Requirement <#18>:Scrambler

Scrambling on the link shall be accomplished through the use of independent transmit and receive scramblers. The scramblers shall be based on the primitive polynomial $x^{48} + x^{28} + x^{27} + x + 1$, and shall have the format shown in Figure F.5.

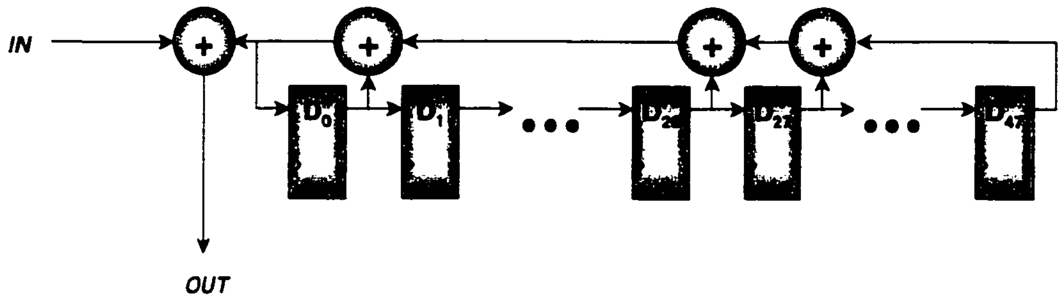


Figure F.5 x^{48} scrambler

Requirement <#19>:Scrambler Operation

The scrambler shall run continuously and shall not be paused during non-scrambled sections of the SDL link data.

Requirement <#20>:Header Scrambling

The 32 bit SDL header ({length,CRC-16} $\oplus$ 0xB6AB31E0) shall not be scrambled.

Requirement <#21>:Payload Scrambling

All bytes of the payload (route tag, payload, payload CRCs) shall be scrambled.

Requirement <#22>:Length Field 0x0001 Special Payload Scrambling

No bytes of the special payload (6 byte message, CRC-16) indicated by length field value 0x0001 shall be scrambled. This special payload is used to convey the scrambler state, and hence can not be scrambled.

Requirement <#23>:Length Field 0x0002 and 0x0003 Special Payload Scrambling

All bytes of the special payloads (6 byte message, CRC-16) indicated by length field values of 0x0002, and 0x0003 shall be scrambled. These special payloads are used for OAM messaging.

F.1.4 Framing

Requirement <#24>:Byte Based

When the SDL link is mapped into a byte aligned transmission structure such as SONET/SDH, framing shall only be done on byte boundaries.

Requirement <#25>:Bit Based

When the SDL link is mapped into a bit based framing structure (i.e. DS-3), or directly onto the transmission medium, framing shall be done on bit boundaries.

Requirement <#26>:Framing State Machine

Framing on an SDL link shall use the following state transition diagram:

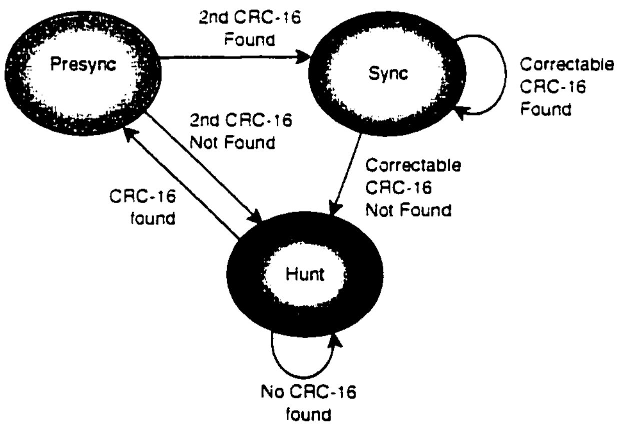


Figure F.6 **SDL framing state machine**

This state transition diagram consists of 3 distinct states: hunt, presync, and sync. In the hunt state, the framer shall, on a bit-by-bit, or byte-by-byte basis search for a sequence of 32 bits that is a valid SDL header (without error correction). Upon finding such a sequence, the framer shall transition to the presync state, and shall determine from the length field and the provisioning information (i.e. route tag size, payload CRC, etc.) how many intervening bytes there are until the next SDL header. When this number of bytes has passed, the framer shall then check the next 32 bits to see if it is also a valid SDL header (again without error correction). If it is not, the SDL framer shall return to the hunt state. On the other hand, if a valid SDL header is found (without error correction), the SDL framer shall move to the sync state and at this point shall be considered in frame. From this point on, single bit error correction on the SDL header shall be enabled.

If at any time in the sync state a correctable SDL header cannot be found where expected, the SDL framer shall transition to the hunt state, and attempt to reacquire frame synchronization.

F.1.5 Scrambler Synchronization

Requirement <#27>:State Transmission

Transmission of the transmitter's scrambler state shall be done using the special payload with the length field value of 0x0001. The 48 bits of scrambler state that are transmitted shall correspond to the state of the scrambler at the MSB of the 48 bit special payload field. An example of a scrambler state packet is shown in Figure F.7.

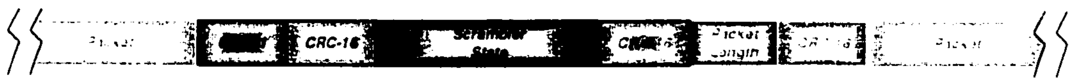


Figure F.7 **Scrambler state header and packet**

Optional Requirement <#28>:State Transmit Interval

It is recommended that the transmit interval of the scrambler state be provisionable on either a per packet basis, or a per byte basis. Currently a value of every 8 packets is recommended, however this value is under study.

Requirement <#29>:Scrambler State Priority

Insertion of the scrambler state shall take precedence over packet transmission (i.e. if it is time to send the scrambler state, and a packet is available, the scrambler state is sent first).

Requirement <#30>:Scrambler Synchronization

Scrambler synchronization shall use the following state transition diagram as shown in Figure F.8. As can be seen there are 3 states: out of frame, sync, and postsync. The out of frame state is the starting point, and where the scrambler state machine stays until the SDL framer has found frame synchronization, and a correctable scrambler state has been received. At this point the scrambler state machine shall load the receive scrambler with the transmitted state, transition to the sync state, and begin passing data.

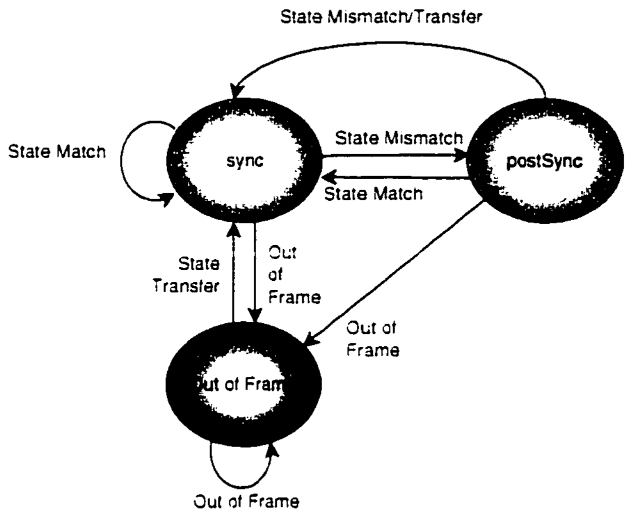


Figure F.8 **Scrambler synchronization state machine**

Upon receiving subsequent correctable scrambler states, the scrambler state machine shall compare the receive scrambler's state with the incoming state. If they match, the scrambler state machine stays in the sync state. If they mismatch, the scrambler state machine shall move to the postsync state, and still pass data. When the next correctable scrambler state is received, this shall be taken as the new reference, and the scrambler state machine shall return to the sync state. If the states mismatched, a synchronization slip will have occurred, whereas if they matched, no error will have occurred.

F.1.6 OAM Messages

Requirement <#31>:A Message

Length field value 0x0002 shall denote an OAM "A" Message, as shown in Figure F.9. Recommended usage of this message is currently under study.

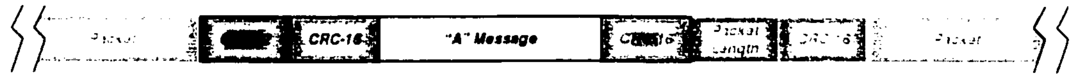


Figure F.9 **"A" Message header and packet**

Requirement <#32>:A Message Priority

Insertion of an A Message shall take precedence over packet transmission, but not over the scrambler state (i.e. if it is time to send the scrambler state, and both an A message and a packet is available, the scrambler state is sent first, followed by the A message, and finally the packet).

Requirement <#33>:B Message

Length field value 0x0003 shall denote an OAM "B" Message, as shown in Figure F.10. Recommended usage of this message is currently under study.

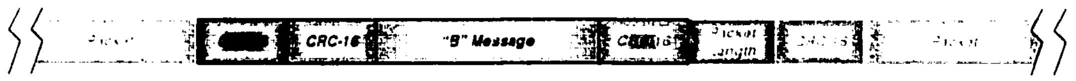


Figure F.10 **"B" Message header**

Requirement <#34>:B Message Priority

Insertion of a B Message shall take precedence over packet transmission, but not over the scrambler state, or an A message.

Appendix G: MATLAB MTTF Program

G.1 Program

```
% the purpose of this is to find the MTTF under varying conditions

clear;

BERStart = -4;
BERStop = -2;
BERIncrement = .1;
L_Avg = 65535;
NMax = 6;

L_CRC = 16;
L_Marker = 32;
L_Max = 16;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BER = 10.^[BERStart:BERIncrement:BERStop];
N = 1:NMax;
MTTF = zeros (length(N), length (BER));
for i = 1:length (N);
for k = 1:length (BER)

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %hunt transitions
 P_bh = 1/(2^L_CRC);
 P_gh = 2*((1-BER(k))^L_Marker) / L_Avg;

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %false presync/sync transitions
 P_hb = (1-P_bh)/(2^(L_Max - 1));
 P_hf = P_hb;
 P_fb = P_bh/(2^(L_Max - 1));
 P_sb = P_gh/(2^(L_Max - 1));
 P_sf = P_sb;

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %good presync/sync transitions
 P_hg = (1-(1-BER(k))^L_Marker) / L_Avg;
 P_hs = P_hg;
 P_sg = P_gh/2;


 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % set up the transition matrix

 % set up the matrix
 sync = 2*N(i)+4;
 t = zeros (sync, sync);
```

```
% hunt transitions
t(2, 1) = 1;
t(2, 3) = P_hb;
t(2, 4) = P_hg;
t(2, sync-1) = P_hf;
%t(2, sync) = P_hs;

if (N(i) > 1)
 % single bad transitions
 t(3, 2) = P_bh;
 t(3, 5) = P_hb;
 t(3, 6) = P_hg;

 % single good transitions
 t(4, 2) = P_gh;
 t(4, 6) = P_hb;

 % multiple bad transitions
 for row = 5:2:(sync-5)
   t(row, row-2) = P_bh;
   t(row, row+2) = P_hb;
   t(row, j+3) = P_hg;
 end

 % multiple good/bad transitions
 for row = 6:2:(sync-4)
   t(row, row-3) = P_gh;
   t(row, row-2) = P_bh;
   t(row, row+2) = P_hb;
 end

 % last multiple bad transitions
 t(sync-3, sync-5) = P_bh;

 % last multiple good/bad transitions
 t(sync-2, sync-5) = P_gh;
 t(sync-2, sync-4) = P_bh;
else
 % single bad transitions: N(i) = 1
 t(3, 2) = P_bh;

 % single good transitions: N(i) = 1
 t(4, 2) = P_gh;
end


% false sync transitions
t(sync-1, 3) = P_fb;
for col = 5:(sync-2)
  t(sync-1, col) = P_fb;
end

% good sync transitions
for col = 3:2:(sync-3)
  t(sync, col) = P_sb;
  t(sync, col+1) = P_sg;
end
t(sync, sync-1) = P_sf;

% go through the diagonal and set the remaining probabilities
d = 1-sum(t);
```

```
  for j = 1:(sync-1)
   t(j, j) = d(j);
  end

  % Find the mean time to frame
  x = zeros (sync, 1);
  x(1, 1) = -1;
  y = inv (t-eye(sync));
  n = -(y^2)*t*x;
  MTTF(i, k) = n(sync);
 end
end

MTTF = MTTF / L_Avg;

figure (1);
set (gcf, 'Color', 'White');
clf;
axes ('Box', 'off');
c = log (MTTF);
caxis ('auto');
mesh (BER, N, MTTF, c);
shading interp;
hold on;
ylabel ('Number of Framers');
xlabel ('BER');
zlabel ('MTTF (packets)');
set (gca, 'XColor', 'Black');
set (gca, 'YColor', 'Black');
set (gca, 'ZColor', 'Black');
set (gca, 'XScale', 'log');
set (gca, 'ZScale', 'log');
set (gcf, 'InvertHardCopy', 'Off');
view (-215, 45);
print -dpsc2 e:\users\paul\text\thesis\sdl\data\mttf.ps;
```

Appendix H: DC Balanced Barker Program

H.1 Program

```
//dcBalancedBarker.cc

#include <complex.h>
#include "standard.h"
//#define DEBUG

class Vector
{
 protected:
   int field, size;
   int* vector;

 public:
   Vector ( ) { size = 0; vector = NULL; }
   Vector (int f, int N);
   ~Vector ( ) { delete [] vector; }

   Vector& operator = (Vector&);
   int& operator [] (int i) { return vector[i]; }
   int operator ++ ( );
};

Vector::Vector (int f, int N)
{
 //declare local variables
 int i;

 field = f; size = N; vector = new (int[N]);
 for (i = 0; i < N; i++) vector[i] = 0;
}

Vector& Vector::operator = (Vector& a)
{
 //declare local variables
 int i;

 if (vector) delete [] vector;

 field = a.field;
 size = a.size;
 for (i = 0; i < size; i++) vector[i] = a.vector[i];

 return (*this);
}


int Vector::operator ++ ( )
```

```
{
 //declare local variables
 int i, carry;

 carry = 0;
 for (i = 0; i < size; i++)
 {
  if (++vector[i] == field)
  {
   vector[i] = 0;
   carry = 1;
  }
  else
  {
   carry = 0;
   break;
  }
 }

 #ifdef DEBUG
 if (vector[3] == 0 &&
    vector[2] == 3 &&
    vector[1] == 1 &&
    vector[0] == 2)
 {
  cout << "We're here\n";
 }
 #endif

 return carry;
}

int main ( )
{
 //declare local variables
 char* function_name = "int main ( )";

 complex *symbol, sum, a, b, dcBalance;
 int fieldSize, N, i, j, k, transitions, mask, nibble;
 double convolution, bestConvolution, real, imaginary, lobeAccumulator, maxLobe;
 String targetFileName = "barker.";
 ofstream target;

 DataLoader infoSource ("barker.info", '#');

 N = infoSource.getInt ( );
 fieldSize = infoSource.getInt ( );

 Vector sequence (fieldSize, N), bestSequence (fieldSize, N);


 //load the map
 symbol = new complex[fieldSize];
 for (i = 0; i < fieldSize; i++)
 {
  real = infoSource.getDouble ( );
  imaginary = infoSource.getDouble ( );
  symbol[i] = complex (real, imaginary);
 }

 //open the target
```

```
targetFileName += String (N);
targetFileName += ".";
targetFileName += String (fieldSize);
target.open (targetFileName);

//write the constellation
target << "# " << N
      << " symbol Barker sequence with the following symbol space:\n";

for (i = 0; i < fieldSize; i++)
{
  target << "# " << i << " = " << symbol[i] << endl;
}
target << "\n\n\n";

//set the starting best convolution
// bestConvolution = norm (symbol[0]);
bestConvolution = 3.0;

maxLobe = 0.0;
for (i = N - 2; i >= 0; i--)
{
  j = N - 1;
  sum = complex (0.0, 0.0);

  #ifdef DEBUG
  cout << "\nLag = " << i << endl << flush;
  #endif

  for (k = i; k >= 0; k--)
  {
    a = symbol[sequence[k]];
    b = conj (symbol[sequence[j]]);

    #ifdef DEBUG
    cout << k << " " << j << "" = "
        << a << " * " << b << " = " << a * b << endl << flush;
    #endif

    sum += a * b;
    j--;
  }

  #ifdef DEBUG
  cout << "Sum = " << sum << endl << flush;
  #endif

  maxLobe += abs (sum);
}

cout << "The sequences with a maximum convolution sidelobe of "
    << bestConvolution << " are:\n\n" << flush;

//find the sequence with lowest excursions
do
{
  //This loop finds the maximum excursion
  //over the convolution for every possible sequence

  //check the current sequence
  convolution = 0.0;
```

```
#ifdef DEBUG
cout << "Current sequence = ";
for (i = N-1; i >= 0; i--) cout << sequence[i];
cout << endl;
#endif


//find the DC balance
dcBalance = 0.0;
for (k = 0; k < N; k++) dcBalance += symbol[sequence[k]];
if (abs (dcBalance)) goto Next;        //bail if no DC balance

//perform the convolution backwards to eliminate as many cases as possible
lobeAccumulator = 0.0;
for (i = N - 2; i >= 0; i--)
{
 j = N - 1;
 sum = complex (0.0, 0.0);

 #ifdef DEBUG
 cout << "\nLag = " << i << endl << flush;
 #endif

 for (k = i; k >= 0; k--)
 {
  a = symbol[sequence[k]];
  b = conj (symbol[sequence[j]]);

  #ifdef DEBUG
  cout << k << " * " << j << " = "
     << a << " * " << b << " = " << a * b << endl << flush;
  #endif

  sum += a * b;
  j--;
 }

 #ifdef DEBUG
 cout << "Sum = " << sum << endl << flush;
 #endif

 lobeAccumulator += abs (sum);
 if (lobeAccumulator > maxLobe) goto Next;  //bail if worse sum

 if (abs (sum) > convolution) convolution = abs (sum);
 if (convolution > bestConvolution) goto Next;
}

if (convolution == bestConvolution)
{
 if (maxLobe > lobeAccumulator) maxLobe = lobeAccumulator;

 for (i = N-1; i >= 0; i--)
 {
  cout << sequence[i];
  target << sequence[i];
 }
 mask = 0x8 >> (7 - ((N-1) % 8));
 cout << " = 0x";
 target << " = 0x";
```

```
      nibble = 0;
      for (i = N-1; i >= 0; i--)
      {
       if (sequence[i]) nibble ^= mask;
       mask >>= 1;
       if (!mask)
       {
        cout << hex << nibble << dec;
        target << hex << nibble << dec;
        mask = 0x8;
        nibble = 0x0;
       }
      }

      //count the number of transitions
      transitions = 0;
      if (sequence[0] != sequence[N-1]) transitions++;   //check back to back
      for (i = 1; i < N; i++) //check internal
       if (sequence[i] != sequence[i-1]) transitions++;

      cout << "   Sum of Lobes = " << lobeAccumulator
          << "  Transitions = " << transitions << endl << flush;
      target << "   Sum of Lobes = " << lobeAccumulator
           << "  Transitions = " << transitions << endl << flush;
      }

     Next: continue;

    } while (!(++sequence));


    delete [] symbol;
    target.close ( );
    return (1);
   }
```

Appendix I: TDAT042G5 Product Brief

Product Brief
March 1999

microelectronics group

Lucent Technologies
Bell Labs Innovations

TDAT042G5 SONET/SDH 155/622/2488 Mb/s

Features

- Versatile IC supports 155/622/2488 Mb/s SONET/SDH interface solutions for SDL/ATM/HDLC applications
- pin compatible with TADM042G5
- Low power 3.3 V supplies.
- compact 600 LBGA package
- -40° C -> +85° C industrial temperature range

SONET/SDH Interface

- Termination of STS-3/STM-1, STS-12/STM-4 or STS-48/STM-16 interface
- Supports overhead processing for all Transport and Path OverHead bytes
- Optional insertion and extraction of overhead bytes via serial overhead interfaces
- Full path termination and SPE extraction/insertion
- SONET/SDH compliant condition and alarm reporting
- Handles all concatenation levels of STS-3c through to STS-48c (in multiples of 3--i.e. 3c, 6c, 9c, etc.)
- Diagnostic loopback modes
- Compliant with Bellcore, ANSI, and ITU standards

Data Processing

- Provisionable back end data framer supports payload extraction of either HDLC, SDL, or ATM (x^{31} or x^{43}) over SONET/SDH as well as PPP packet encapsulation (CRC-16/32)
- also supports SDL/ATM directly over fiber
- supports x^{43} prescrambling and postscrambling options for HDLC
- Integrated standard UTOPIA Level 2 and 3 interface for ATM cells
- Enhanced UTOPIA interface for packets
- Maintains counts for cell/packet traffic (including total number of cells, number of discarded cells)
- Compliant with ATM Forum, IETF, and ITU specifications

Microprocessor Interface

- 16 bit address with 16 bit data interface and up to 50 MHz read and write access
- compatible with most industry standard microprocessors

Description

The TDAT042G5 SONET/SDH interface provides a versatile interface for all OC-3/OC-12/OC-48 point-to-point datacom and telecom applications. Constructed using Lucent's state of the art 0.25µ CMOS technology this chip has integrated SONET/SDH framing, Path Termination and data engine blocks.

Communication with the TDAT042G5 is accomplished through a generic 16-bit microprocessor interface. The device supports both multiplexed and separate address and data buses.

The TDAT042G5 with an integrated data processor provides the option of either SDL, ATM, or HDLC framing within the standard SONET/SDH SPE payload, or for SDL and ATM directly over fiber.

With the TDAT042G5, construction of all types of point-to-point OC-3/OC-12/OC-48 data equipment is simplified and cost reduced allowing extremely competitive solutions to be constructed.

The TDAT042G5 SONET/SDH interface provides a simple, completely ATM Forum/ITU compliant interface for all OC-3/OC-12/OC-48 applications.

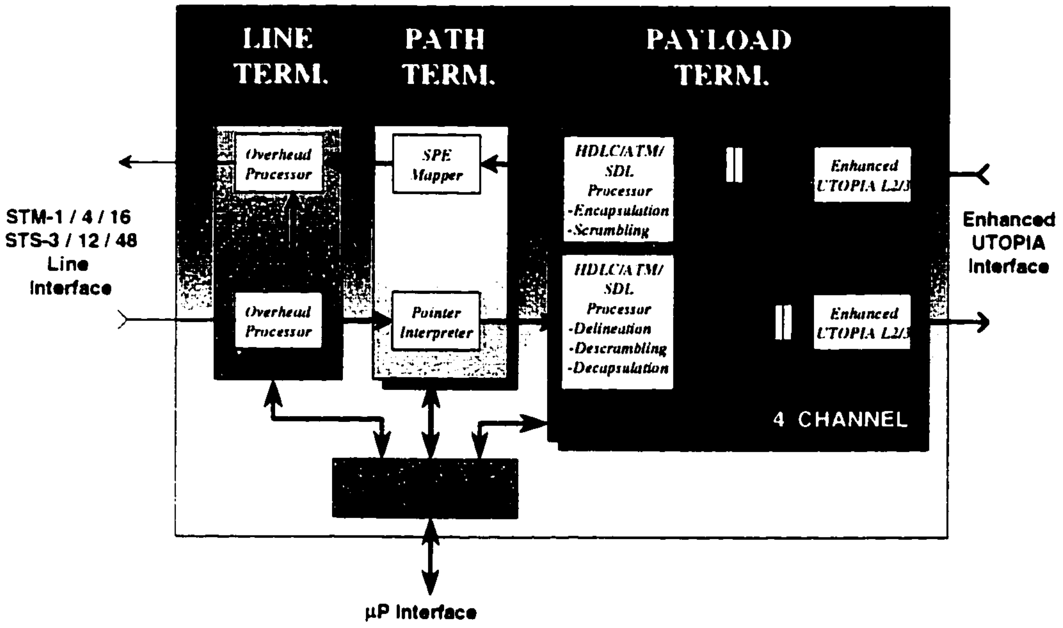

LINE TERM.
PATH TERM.
PAYLOAD TERM.
Overhead Processor
SPE Mapper
HDLC/ATM/ SDL Processor -Encapsulation -Scrambling
Enhanced UTOPIA L2/3
STM-1 / 4 / 16 STS-3 / 12 / 48 Line Interface
Enhanced UTOPIA Interface
Overhead Processor
Pointer Interpreter
HDLC/ATM/ SDL Processor -Delineation -Descrambling -Decapsulation
Enhanced UTOPIA L2/3
4 CHANNEL
µP Interface

Figure 1 Block Diagram of TDAT042G5

For additional information, contact your Microelectronics Group Account Manager or the following:
INTERNET: **http://www.lucent.com/micro**
U.S.A.: Microelectronics Group, Lucent Technologies Inc., 555 Union Boulevard, Room 30L-15P-BA, Allentown, PA 18103
1-800-372-2447, FAX 610-712-4106 (In CANADA: **1-800-553-2448**, FAX 610-712-4106), e-mail
docmaster@micro.lucent.com
ASIA PACIFIC: Microelectronics Group, Lucent Technologies Singapore Pte. Ltd., 77 Science Park Drive, #03-18 Cintech III, Singapore 118256
Tel. (65) 778 8833, FAX (65) 777 7495
JAPAN: Microelectronics Group, Lucent Technologies Japan Ltd., 7-18, Higashi-Gotanda 2-chome, Shinagawa-ku, Tokyo 141, Japan
Tel. (81) 3 5421 1600, FAX (81) 3 5421 1700
For data requests in Europe:
MICROELECTRONICS GROUP DATALINE: **Tel. (44) 1189 324 299**, FAX (44) 1189 328 148
For technical inquiries in Europe:
CENTRAL EUROPE: **(49) 89 95086 0** (Munich), NORTHERN EUROPE: **(44) 1344 865 900** (Bracknell UK),
FRANCE: **(33) 1 41 45 77 00** (Paris), SOUTHERN EUROPE: **(39) 2 6601 1800** (Milan) or **(34) 1 807 1700** (Madrid)

March 1999

Printed On Recycled Paper

microelectronics group

Lucent Technologies
Bell Labs Innovations

Appendix J: SDL IETF Draft

J.1 Draft

```
PPP Working Group                                   J. Carlson
Internet Draft                                 IronBridge Networks
Expires December 1999                                P. Langner
                  Lucent Technologies Microelectronics Group
                                        E. J. Hernandez-Valencia
                                           Lucent Technologies
                                                 J. Manchester
                                           Lucent Technologies
                                                     June 1999
```

PPP over Simple Data Link (SDL)
using SONET/SDH with ATM-like framing
<draft-ietf-pppext-sdl-02.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

To view the list Internet-Draft Shadow Directories, see http://www.ietf.org/shadow.html.

This document is the product of the Point-to-Point Protocol Extensions Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the ietf-ppp@merit.edu mailing list.

Distribution of this memo is unlimited.

Abstract

The Point-to-Point Protocol (PPP) [1] provides a standard method for transporting multi-protocol datagrams over point-to-point links, and RFCs 1662 [2] and 1619 [3] provide a means to carry PPP over Synchronous Optical Network (SONET) [4] and Synchronous Digital Hierarchy (SDH) [5] circuits. This document extends these standards to include a new encapsulation for PPP called Simple Data Link (SDL) [6]. SDL provides a very low overhead alternative to standard HDLC encapsulation, and can also be used on SONET/SDH links.

Applicability

This specification is intended for those implementations that use PPP over high speed point-to-point circuits, both with so-called "dark fiber" and over public telecommunications networks. Because this enhanced PPP encapsulation has very low overhead and good hardware scaling characteristics, it is anticipated that significantly higher throughput can be attained when compared to other possible SONET/SDH payload mappings, and at a significantly lower cost for line termination equipment.

SDL is defined over other media types and for other data link protocols, but this specification covers only the use of PPP over SDL on SONET/SDH.

The use of SDL requires the presentation of packet length information in the SDL header. Thus, hardware implementing SDL must have access to the packet length when generating the header, and where a router's input link does not have this information (that is, for non-SDL input links), the router may be required to buffer the entire packet before transmission. "Worm-hole" routing is thus at least problematic with SDL, unless the input links are also SDL. This, however, does not appear to be a great disadvantage on modern routers due to the general requirement of length information in other parts of the system, notably in queuing and congestion control strategies such as Weighted Fair Queuing [7] and Random Early Detect [8].

Table of Contents

1. Introduction

The Path Signal Label (SONET/SDH overhead byte named C2; referred to as PSL in this document) is intended to indicate the type of data carried on the path. This data, in turn, is referred to as the SONET Synchronous Payload Envelope (SPE) or SDH Administrative Unit Group (AUG). The experimental PSL value of decimal 207 (CF hex) is currently [3] used to indicate that the SPE contains PPP framed using RFC 1662 Octet Synchronous (O-S) framing and transmission without scrambling, and the value 22 (16 hex) is used to indicated PPP framed using O-S framing and transmission with ATM-style X^43+1 scrambling.

This document describes a method to enable the use of SDL framing for PPP over SONET/SDH, and describes the framing technique and requirements for PPP. While O-S framing has a worst-case octet overhead of 100% of all data octets transmitted, SDL has a fixed eight octet per frame overhead with zero data overhead. This mapping is similar to the earlier "Ether-like Framing" proposal, found in a separate work-in-progress. [9]

Note: This document describes a new SONET/SDH PSL value 23 (17 hex). This value has not been allocated by any applicable standards body, and SDL must not be used on public networks until a standard value is allocated. A joint contribution will be made to ANSI subcommittee T1X1.5 requesting the assignment of 17 hex as a PSL for an SDL over SONET mapping in T1.105.

2. Physical Layer Requirements

PPP treats SONET/SDH transport as octet-oriented synchronous links. No provision is made to transmit partial octets. Also, SONET/SDH links are full-duplex by definition.

2.1. Payload Types

Only synchronous payloads STS-1 and higher are considered in this document. Lower speed synchronous, such as VT1.5-SPE/VC-11, and plesiochronous payload mappings, such as T1 and T3, are defined for SONET/SDH and for the SDL algorithm itself, but, since standard HDLC is defined for PPP on those media, PPP over SDL is not defined.

SDL is separately defined as a PPP transport for use on raw fiber without SONET/SDH framing for use as an alternative to bit-synchronous HDLC. Please see the separate work-in-progress for details.

2.2. Control Signals

The PPP over SONET/SDH mapping allows the use of the PSL as a control signal. Not all equipment, however, is capable of setting or detecting this value, and any standard must take this into account. Equipment employing only SDL MUST be capable of transmitting PSL with value 23, but need not be capable of detecting the peer's value or capable of changing its own value.

There are two methods to enable SDL, an LCP-negotiated method and a prior-arrangement method. The former allows for easier configuration and compatibility with existing equipment, while the latter allows general use with separate SONET/SDH transmission equipment with PSL limitations. Both types of implementations will freely interoperate given the procedures below.

LCP-negotiated systems MUST be capable of changing their transmitted PSL value and detecting the peer's value. Equipment without these features MUST NOT support LCP negotiation of SDL.

When SDL is negotiated by LCP, LCP negotiation MUST be started with the PSL value initially set to 22 or 207 and the corresponding non-SDL O-S PPP encapsulation MUST be used. The SDL LCP option is then placed in the LCP Configure-Request messages transmitted. On reception of LCP Configure-Request with an SDL LCP option or when the peer's transmitted PSL value is received as 23, the implementation MUST shut down LCP by sending a Down event to its state machine, then switch its transmitted PSL value to 23, switch encapsulation mode to SDL, wait for SDL synchronization, and then restart LCP by sending an Up event into LCP. Otherwise, if the peer does not transmit PSL value 23 and it does not include the SDL LCP option in its LCP Configure-Request messages, then operation using non-SDL O-S PPP encapsulation continues. If the received PSL value subsequently received reverts from 23 to any other value, then this is treated as a Down event into the LCP state machine, and an Up event MUST be generated if the new value is recognized as a valid PPP framing mode.

When SDL is enabled by prior arrangement, the PSL SHOULD be transmitted as 23. Any other value may also be used by prior external arrangement with the peer, although the values 22 and 207 are discouraged. (Such use is enforced by an administrator, and is outside the scope of this specification.) When SDL is enabled by prior arrangement, the SDL LCP option SHOULD NOT be negotiated by the peers.

An implementation-specific configuration option SHOULD exist to enable the use of prior-arrangement versus LCP-negotiated modes. This option SHOULD be presented to an administrator, and SHOULD default to LCP-negotiated if the hardware permits. Otherwise, if the hardware implementation precludes non-SDL modes of operation, then it MUST default to prior-arrangement mode.

2.3. Synchronization Modes

Unlike O-S encapsulation, SDL provides a positive indication that it has achieved synchronization with the peer. An SDL PPP implementation MUST provide a means to temporarily suspend PPP data transmission (both user data and negotiation traffic) if synchronization loss is detected. An SDL PPP implementation SHOULD also provide a configurable timer that is started when SDL is initialized and restarted on the loss of synchronization, and is terminated when link synchronization is achieved. If this timer expires, implementation-dependent action should be taken to report the hardware failure.

2.4. Simple-Data-Link LCP Option

A new LCP Configuration Option is used to request Simple Data Link (SDL) [6] operation for the PPP link.

A summary of the Simple-Data-Link Configuration Option format for the Link Control Protocol (LCP) is shown below. The fields are transmitted from left to right.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

29

Length

2

This option is used only as a hint to the peer that SDL operation is preferred by the sender. If the current encapsulation mode is not SDL, then the only appropriate response to reception of this option by an SDL speaker is to then switch the encapsulation mode to SDL (as detailed in the section above) and restart LCP. Non SDL-speakers SHOULD instead send LCP Configure-Reject for the option.

If either LCP Configure-Nak or LCP Configure-Reject is received for this option, then the next transmitted LCP Configure-Request MUST NOT include this option. If LCP Configure-Ack with this option is received, it MUST NOT be treated as a request to switch into SDL mode. If the received LCP Configure-Request message does not contain an SDL LCP option, an implementation MUST NOT send an unsolicited Configure-Nak for the option.

(An implementation of SDL that is already in SDL framing mode and receives this option in an LCP Configure-Request message MAY, both for clarity and for convergence reasons, elect to send LCP Configure-Ack. It MUST NOT restart LCP nor change framing modes in this case.)

2.5. Framing

The PPP frames are located by row within the SPE payload. Because frames are variable in length, the frames are allowed to cross SPE boundaries. Bytes marked as "overhead" or "fixed stuff" in SONET/SDH documentation for concatenated streams are not used as payload bytes.

When SDL framing for PPP is employed, the SDL "Datagram Offset" is fixed at 4, and the "A" and "B" messages are never used. These optional features of SDL are not described in this document, but are rather described in Lucent's SDL specification [6].

Fixing the Datagram Offset to 4 allows a PPP MRU/MTU of 65536 using SDL.

SDL framing is in general accomplished by the use of a four octet header on the packet. This fixed-length header allows the use of a simple framer to detect synchronization as described in section 3.6. For use with PPP, this header precedes each raw PPP packet as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Packet Length         |          Header CRC           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   PPP packet (beginning with address and control fields)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             .....                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Packet CRC                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The four octet length header is DC balanced by exclusive-OR (also known as "modulo 2 addition") with the hex value B6AB31E0. This is the maximum transition, minimum sidelobe, Barker-like sequence of length 32. No other scrambling is done on the header itself.

Packet Length is an unsigned 16 bit number in network byte order. Unlike the standard PPP FCS, the Header CRC is a CRC-16 generated with initial value zero and transmitted in standard network byte order. The PPP packet is scrambled, and begins with the standard address and control fields, and may be any integral octet length (i.e., it is not padded unless the Self Describing Padding option is used). The Packet CRC is also scrambled, and has a mode-dependent length (described below), and is located only on an octet boundary; no alignment of this field may be assumed.

When the Packet Length value is 4 or greater, the distance in octets between one message header and the next in SDL is the sum of Packet Length field, Datagram Offset value, and the fixed size of the Packet CRC field. The Datagram Offset is a configurable SDL parameter, which is set to the fixed value 4 for PPP. When the Packet Length is 0, the distance to the next header is 4 octets. This is the idle fill header. When the Packet Length is 1 to 3, the distance to the next header is 12 octets. These headers are used for special SDL messages used only with optional scrambling and management modes. See section 5 for details of the messages.

General SDL, like PPP, allows the use of no CRC, ITU-T CRC-16, or ITU-T CRC-32 for the packet data. However, because the Packet Length field does not include the CRC length, synchronization cannot be maintained if the CRC type is changed per RFC 1570, because frame-to-frame distance is, as described above, calculated including the CRC length. Thus, this PPP over SDL specification fixes the CRC type to CRC-32 (four octets), and all SDL implementations MUST reject any LCP FCS Alternatives Option [10] requested by the peer when in SDL mode.

PPP over SDL implementations MAY allow a configuration option to set different CRC types for use by prior arrangement. Any such configurable option MUST default to CRC-32, and MUST NOT be include LCP negotiation of FCS Alternatives.

With the SDL Datagram Offset set to 4, the value placed in the Packet Length field is exactly the length in octets of the PPP frame itself, including the address and control fields but not including the FCS field.

Because Packet Lengths below 4 are reserved, the Packet Length MUST be 4 or greater for any legal PPP packet. PPP packets with fewer octets, which are not possible without address/control or protocol field compression, MUST be padded to length 4 for SDL.

Inter-packet time fill is accomplished by sending the four octet length header with the Packet Length set to zero. No provision is made for intra-packet time fill.

By default, an independent, self-synchronous x^43+1 scrambler is used on the data portion of the message including the 32 bit CRC. This is done in exactly the same manner as with the ATM x^43+1 scrambler on an ATM channel with exactly one VCC. The scrambler is not clocked when SDL header bits are transmitted. Thus, the data scrambling can be implemented in an entirely independent manner from the SDL framing.

Optionally, by prior arrangement, SDL links MAY use a set-reset scrambler as described in section 6. If this option is provided, it MUST be configurable by the administrator, and the option MUST default to the self-synchronous scrambler.

2.6. Synchronization Procedure

The link synchronization procedure is similar to the I.432 section 4.5.1.1 ATM HEC delineation procedure [11], but simpler because the SDL messages are variable length. The machine starts in HUNT state until a four octet sequence in the data stream with a valid CRC-16 is found. (Note that the CRC-16 single-bit error correction technique described in section 3.9 is not employed until the machine is in in SYNCH state. The header must have no bit errors in order to leave HUNT state.) Such a valid sequence is a candidate SDL header. On finding the valid sequence, the machine enters PRESYNCH state. Any one invalid SDL header in PRESYNCH state returns the link to HUNT state.

If a second valid SDL header is seen after entering PRESYNCH state, then the link enters SYNCH state and PPP transmission is enabled. If an invalid SDL header is detected, then the link is returned to HUNT state without enabling PPP transmission.

Once the link enters SYNCH state, the SDL header single bit error correction logic is enabled (see section 3.9). Any unrecoverable header CRC error returns the link to HUNT state, disables PPP transmission, and disables the error correction logic.

2.7. Scrambler Operation

The transmit and receive scramblers are shift registers with 43 stages that MAY be initialized to all-ones when the link is initialized. Synchronization is maintained by the data itself.

```
        Transmit                      Receive

 DATA-STREAM (FROM PPP)          IN (FROM SDL FRAMER)
 |                    |
 v                    |
 XOR<-------------------------+   +->D0-+->D1-> ... ->D41->D42-+
 |                  |   |                 |
 +->D0-+->D1-> ... ->D41->D42-+   XOR<-------------------------+
 |                    |
 v                    v
 OUT (TO SDL FRAMER)          DATA-STREAM (TO PPP)
```

Each XOR is an exclusive-or gate; also known as a modulo-2 adder. Each Dn block is a D-type flip-flop clocked on the appropriate data clock.

The scrambler is clocked once after transmission or reception of each bit of payload and before the next bit is applied as input. Bits within an octet are, per SONET/SDH standard practice, transmitted and received MSB-first.

2.8. CRC Generation

The CRC-16 and CRC-32 generator polynomials used by SDL are the ITU-T standard polynomials [12]. These are:

x^16+x^12+x^5+1

x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1

The SDL Header CRC and the CRC-16 used for each of the three special messages (scrambler state, message A, and message B; see section 5) are all generated using an initial remainder value of 0000 hex.

The optional CRC-16 on the payload data (this mode is not used with PPP over SDL except by prior arrangement) uses the standard initial remainder value of FFFF hex for calculation and the bits are complemented before transmission. The final CRC remainder, however, is transmitted in network byte order, unlike the regular PPP FCS. If the CRC-16 algorithm is run over all of the octets including the appended CRC itself, then the remainder value on intact packets will always be E2F0 hex. Alternatively, an implementation may stop CRC calculation before processing the appended CRC itself, and do a direct comparison.

The standard CRC-32 on the payload data (used for standard PPP over SDL) uses the initial remainder value of FFFFFFFF hex for calculation and the bits are complemented before transmission. The CRC, however, is transmitted in network byte order, most significant bit first, unlike the optional PPP 32 bit FCS, which is transmitted in reverse order. The remainder value on intact packets when the appended CRC value is included in the calculation is 38FB2284.

C code to generate these CRCs is found in Appendix A.

2.9. Error Correction

The error correction technique is based on the use of a Galois number field, as with the ATM HEC correction. In a Galois number field, f(a+b) = f(a) + f(b). Since the CRC-16 used for SDL forms such a field, we can state that CRC(message+error) = CRC(message) + CRC(error). Since the CRC-16 remainder of a properly formed message is always zero, this means that, for the N distinct "error" strings corresponding to a single bit error, there are N distinct CRC(error) values, where N is the number of bits in the message.

A table look-up is thus applied to the CRC-16 residue after calculation over the four octet SDL header to correct bit errors in the header and to detect multiple bit errors. For the optional set-reset scrambler, a table look-up is similarly applied to the CRC-16 residue after calculation over the eight octet scrambler state message to correct bit errors and to detect multiple bit errors. (This second correction is also used for the special SDL A and B messages, which are not used for standard PPP over SDL.)

Note: No error correction is performed for the payload.

Note: This error correction technique is used only when the link has entered SYNCH state. While in HUNT or PRESYNCH state, error correction should not be performed, and only messages with syndrome 0000 are accepted. If the calculated syndrome does not appear in this table, then an unrecoverable error has occurred. Any such error in the SDL header will return the link to HUNT state.

Since the CRC calculation is started with zero, the two tables can be merged. The four octet table is merely the last 32 entries of the eight octet table.

Eight octet (64 bit) single bit error syndrome table (in hex):

```
FD81 F6D0 7B68 3DB4 1EDA 0F6D 8FA6 47D3
ABF9 DDEC 6EF6 377B 93AD C1C6 60E3 B861
D420 6A10 3508 1A84 0D42 06A1 8B40 45A0
22D0 1168 08B4 045A 022D 8906 4483 AA51
DD38 6E9C 374E 1BA7 85C3 CAF1 ED68 76B4
3B5A 1DAD 86C6 4363 A9A1 DCC0 6E60 3730
1B98 0DCC 06E6 0373 89A9 CCC4 6662 3331
9188 48C4 2462 1231 8108 4084 2042 1021
```

Thus, if the syndrome 6EF6 is seen on an eight octet message, then the third bit (hex 20) of the second octet is in error. Similarly, if 48C4 is seen on an eight octet message, then the second bit (hex 40) in the eighth octet is in error. For a four octet message, the same two syndromes would indicate a multiple bit error for 6EF6, and a single bit error in the second bit of the fourth octet for 48C4.

Note that eight octet messages are used only for the optional set-reset scrambling mode, described in section 6.

Corresponding C code to generate this table is found in Appendix B.

3. Performance Analysis

There are five general statistics that are important for framing algorithms. These are:

MTTF	Mean time to frame
MTTS	Mean time to synchronization
PFF	Probability of false frame
PFS	Probability of false synchronization
PLF	Probability of loss of frame

The following sections summarize each of these statistics for SDL. Details and mathematic development can be found in the Lucent SDL documentation [6].

3.1. Mean Time To Frame (MTTF)

This metric measures the amount of time required to discover correct framing in the input data. This may be measured in any convenient units, such as seconds or bytes. For SDL, the relevant measurement is in packets, since fragments of packets are not useful.

In order to calculate MTTF, we must first determine how often the frame detection state machine is "unavailable" because it has detected an apparent framing header within the user data.

Since the probability of a false header detection using CRC-16 in random data is 2^-16 and this rate is large compared to the allowable packet size, it is worthwhile to run multiple parallel frame-detection state machines. Each machine starts with a different candidate framing point in order to reduce the probability of falsely detecting user data as a valid frame header.

The results for this calculation, given maximal 64KB packets and average 384 byte packets, are:

Number of Framers	Unavailability 64KB packets	Unavailability 384 byte pkts
1	367.9E-3	5.373E-3
2	30.83E-3	1.710E-6
3	2.965E-3	971.2E-12
4	253.2E-6	465.3E-15

Using these values, MTTF can be calculated as a function of the Bit Error Rate (BER). These plots show a characteristically flat region for all BERs up to a knee, beyond which the begins to rise sharply. In all cases, this knee point has been found to occur at a BER of approximately 1E-4, which is several orders of magnitude above that observed on existing SONET/SDH links. The flat rate values are summarized as:

Number of Framers	Flat region 64KB packets	Flat region 384 bytes
1	3.58	1.52
2	1.595	1.5
3	1.52	1.5
4	1.5	1.5

Thus, for common packet sizes in an implementation with two parallel framers using links with a BER of 1E-4 or better, the MTTF is approximately 1.5 packets. This is also the optimal time, since it represents initiating framing at an average point half-way into one packet, and achieving good framing after seeing exactly one correctly framed packet.

3.2. Mean Time To Synchronization (MTTS)

The MTTS for standard SDL with a self-synchronous scrambler is the same as the MTTF, or 1.5 packets.

The MTTS for SDL using the optional set-reset scrambler is one half of the scrambling state transmission interval (in packets) plus the MTTF. For insertion at the default rate of one per eight packets, the MTTS is 5.5 packets.

(The probability of receiving a bad scrambling state transmission should also be included in this calculation. The probability of random corruption of this short message is shown in the SDL document [6] to be small enough that it can be neglected for this calculation.)

3.3. Probability of False Frame (PFF)

The PFF is 232.8E-12 (2^-32), since false framing requires two consecutive headers with falsely correct CRC-16.

3.4. Probability of False Synchronization (PFS)

The PFS for the standard self-synchronous scrambler is the same as the PFF, or 232.8E-21 (2^-32).

The PFS for the set-reset scrambler is 54.21E-21 (2^-64), and is calculated as the PFF above multiplied by the probability of a falsely detected scrambler state message, which itself contains two independent CRC-16 calculations.

3.5. Probability of Loss of Frame (PLS)

The PLS is a function of the BER, and for SDL is approximately BER multiplied by .005, which is the probability of two or more bit errors occurring within the 32 bit SDL header. Thus, at a BER of 1E-5, the PLS is 5E-8.

4. The Special Messages

When the SDL Packet Length field has any value between 0000 and 0003, the message following the header has a special, pre-defined length. The 0 value is a time-fill on an idle link, and no other data follows. The next octet on the link is the first octet of the next SDL header.

The values 1 through 3 are defined in the following subsections. These special messages each consist of a six octet data portion followed by another CRC-16 over that data portion, as with the SDL header, and this CRC is used for single bit error correction.

4.1. Scrambler State

The special value of 1 for Packet Length is reserved to transfer the scrambler state from the transmitter to the receiver for the optional set-reset scrambler. In this case, the SDL header is followed by six octets (48 bits) of scrambler state. Neither the scrambler state nor the CRC are scrambled.

4.2. A/B Message

The special values of 2 and 3 for Packet Length are reserved for "A" and "B" messages, which are also six octets in length followed by two octets of CRC-16. Each of these eight octets are scrambled. No use for these messages with PPP SDL is defined. These messages are reserved for use by link maintenance protocols, in a manner analogous to ATM's OAM cells.

5. The Set-Reset Scrambler Option

Standard PPP over SDL uses a self-synchronous scrambler. SDL implementations MAY also employ a set-reset scrambler to avoid some of the possible inherent problems with self-synchronous scramblers.

5.1. The Killer Packet Problem

Scrambling in general solves two problems. First, SONET and SDH interfaces require a minimum density of bit transitions in order to maintain hardware clock recovery. Since data streams frequently contain long runs of all zeros or all ones, scrambling the bits using a pseudo-random number sequence breaks up these patters. Second, all link-layer synchronization mechanisms rely on detecting long-range patterns in the received data to detect framing.

Self-synchronous scramblers are an easy way to partially avoid these problems. One problem that is inherent with self-synchronous, however, is that long user packets from malicious sites can make use of the known properties of these scramblers to inject either long strings of zeros or other synchronization-destroying patterns into the link. For public networks, where the data presented to the network is usually multiplexed (interleaved) with multiple unrelated streams, the clocking problem does not pose a significant threat to the public network. It does, however, pose a threat to the PPP-speaking device, and it poses a threat to long lines that are unchannelized.

Such carefully constructed packets are called "killer packets."

5.2. SDL Set-Reset Scrambler

An alternative to the self-synchronous scrambler is the externally synchronized or "set-reset" scrambler. This is a free-running scrambler that is not affected by the patterns in the user data, and therefore minimizes the possibility that a malicious user could present data to the network that mimics an undesirable data pattern.

The option set-reset scrambler defined for SDL is an x^48+x^28+x^27+x+1 independent scrambler initialized to all ones when the link enters PRESYNCH state and reinitialized if the value ever becomes all zero bits. As with the self-synchronous scrambler, all octets in the PPP packet data following the SDL header through the final packet CRC are scrambled.

5.3. SDL Scrambler Synchronization

As described in the previous section, the special value of 1 for Packet Length is reserved to transfer the scrambler state from the transmitter to the receiver. In this case, the SDL header is followed by six octets (48 bits) of scrambler state plus two octets of CRC-16 over the scrambler state. None of these eight octets are scrambled.

SDL synchronization consists of two components, link and scrambler synchronization. Both must be completed before PPP data flows on the link.

If a valid SDL header is seen in PRESYNCH state, then the link enters SYNCH state, and the scrambler synchronization sequence is started. If an invalid SDL header is detected, then the link is returned to HUNT state, and PPP transmission is suspended.

When scrambler synchronization is started, a scrambler state message is sent (Packet Length set to 1 and six octets of scrambler state in network byte order follow the SDL header). This message is sent once. At this point, PPP transmission is enabled.

Scrambler state messages are periodically transmitted to keep the peers in synchronization. A period of once per eight transmitted packets is suggested, and it SHOULD be configurable. Excessive packet CRC errors detected indicates an extended loss of synchronization and should trigger link resynchronization.

On reception of a scrambler state message, an SDL implementation MUST compare the received 48 bits of state with the receiver's scrambler state. If any of these bits differ, then a synchronization slip error is declared. After such an error, the next valid scrambler state message received MUST be loaded into the receiver's scrambler, and the error condition is then cleared.

5.4. SDL Scrambler Operation

The transmit and receive scramblers are shift registers with 48 stages that are initialized to all-ones when the link is initialized. Each is refilled with all one bits if the value in the shift register ever becomes all zeros. This scrambler is not reset at the beginning of each frame, as is the SONET/SDH X^7+X^6+1 scrambler, nor is it modified by the transmitted data, as is the ATM self-synchronous scrambler. Instead it is kept in synchronization using special SDL messages.

```
+----XOR<-------------XOR<---XOR<---------------+
|     ^               ^     ^                   |
|     |               |     |                   |
+-->D0-+-->D1-> ... ->D26-+-->D27-+-->D28-> ... ->D47-+
|
v
OUT
```

Each XOR is an exclusive-or gate; also known as a modulo-2 adder. Each Dn block is a D-type flip-flop clocked on the appropriate data clock.

The scrambler is clocked once after transmission of each bit of SDL data, whether or not the transmitted bit is scrambled. When scrambling is enabled for a given octet, the OUT bit is exclusive-ored with the raw data bit to produce the transmitted bit. Bits within an octet are transmitted MSB-first.

Reception of scrambled data is identical to transmission. Each received bit is exclusive-ored with the output of the separate receive data scrambler.

To generate a scrambler state message, the contents of D47 through D0 are snapshot at the point where the first scrambler state bit is sent. D47 is transmitted as the first bit of the output. The first octet transmitted contains D47 through D40, the second octet D39 through D32, and the sixth octet D7 through D0.

The receiver of a scrambler state message MUST first run the CRC-16 check and correct algorithm over this message. If the CRC-16 message check detects multiple bit errors, then the message is dropped and is not processed further.

Otherwise, it then should compare the contents of the entire receive scrambler state D47:D0 with the corrected message. (By pipelining the receiver with multiple clock stages between SDL Header error-correction block and the descrambling block, the receive descrambler will be on the correct clock boundary when the message arrives at the descrambler. This means that the decoded scrambler state can be treated as immediately available at the beginning of the D47 clock cycle into the receive scrambler.)

If any of the received scrambler state bits is different from the corresponding shift register bit, then a soft error flag is set. If the flag was already set when this occurs, then a synchronization slip error is declared. This error SHOULD be counted and reported through implementation-defined network management procedures. When the receiver has this soft error flag set, any scrambler state message that passes the CRC-16 message check without multiple bit errors is clocked directly into the receiver's state register after the comparison is done, and the soft error flag is then cleared. Otherwise, while uncorrectable scrambler state messages are received, the soft error flag state is maintained.

(The intent of this mechanism is to reduce the likelihood that a falsely corrected scrambler state message with multiple bit errors can corrupt the running scrambler state.)

6. Configuration Details

6.1. Default LCP Configuration

The standard LCP synchronous configuration defaults apply to SONET/SDH links.

The following Configuration Options are recommended:

Magic Number
No Address and Control Field Compression
No Protocol Field Compression
No FCS alternatives (32-bit FCS default)

This configuration means that standard PPP over SDL generally presents a 32-bit aligned datagram to the network layer. With the address, control, and protocol field intact, the PPP overhead on each packet is four octets. If the SDL framer presents the SDL packet header to the PPP input handling in order to communicate the packet length (the Lucent implementation does not do this, but other hardware implementations may), this header is also four octets, and alignment is preserved.

6.2. Modification of the Standard Frame Format

Since SDL does take the place of HDLC as a transport for PPP, it is at least tempting to remove the HDLC-derived overhead. This is not done for standard PPP over SDL in order to preserve the message alignment and to allow for the future possibility interworking with other services (e.g., Frame Relay).

By prior external arrangement or via standard LCP negotiation, any two SDL implementations MAY agree to omit the address and control fields or implement protocol field compression on a link. Such use is not standardized and MUST NOT be the default on any SDL implementation.

Appendix A: CRC Generation

The following unoptimized code generates proper CRC-16 and CRC-32 values for SDL messages. Note that the polynomial bits are numbered in big-endian order for SDL CRCs; bit 0 is the MSB.

```
typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned long u32;

#define POLY16  0x1021
#define POLY32  0x04C11DB7

u16
crc16(u16 crcval, u8 cval)
{
   int i;

   crcval ^= cval << 8;
   for (i = 8; i--; )
      crcval = crcval & 0x8000 ? (crcval << 1) ^ POLY16 :
         crcval << 1;
   return crcval;
}

u32
crc32(u32 crcval, u8 cval)
{
   int i;

   crcval ^= cval << 24;
   for (i = 8; i--; )
      crcval = crcval & 0x80000000 ? (crcval << 1) ^ POLY32 :
         crcval << 1;
   return crcval;
}

u16
crc16_special(u8 *buffer, int len)
{
   u16 crc;

   crc = 0;
   while (--len >= 0)
      crc = crc16(crc,*buffer++);
   return crc;
}
```

```
u16
crc16_payload(u8 *buffer, int len)
{
  u16 crc;

  crc = 0xFFFF;
  while (--len >= 0)
    crc = crc16(crc,*buffer++);
  return crc ^ 0xFFFF;
}

u32
crc32_payload(u8 *buffer, int len)
{
  u32 crc;

  crc = 0xFFFFFFFFul;
  while (--len >= 0)
    crc = crc32(crc,*buffer++);
  return crc ^ 0xFFFFFFFFul;
}

void
make_sdl_header(int packet_length, u8 *buffer)
{
  u16 crc;

  buffer[0] = (packet_length >> 8) & 0xFF;
  buffer[1] = packet_length & 0xFF;
  crc = crc16_special(buffer,2);
  buffer[0] ^= 0xB6;
  buffer[1] ^= 0xAB;
  buffer[2] = ((crc >> 8) & 0xFF) ^ 0x31;
  buffer[3] = (crc & 0xFF) ^ 0xE0;
}
```

Appendix B: Error Correction Tables

To generate the error correction table, the following implementation may be used. It creates a table called sdl_error_position, which is indexed on CRC residue value. The tables can be used to determine if no error exists (table entry is equal to FE hex), one correctable error exists (table entry is zero-based index to errored bit with MSB of first octet being 0), or more than one error exists, and error is uncorrectable (table entry is FF hex). To use for eight octet messages, the bit index from this table is used directly. To use for four octet messages, the index is treated as an unrecoverable error if it is below 32, and as bit index plus 32 if it is above 32.

The program also prints out the error syndrome table shown in section 3.9. This may be used as part of a "switch" statement in a hardware implementation.

```
u8 sdl_error_position[65536];

/* Calculate new CRC from old^(byte<<8) */
u16
crc16_t8(u16 crcval)
{
  u16 f1,f2,f3;

  f1 = (crcval>>8) | (crcval<<8);
  f2 = (crcval>>12) | (crcval&0xF000) | ((crcval>>7)&0x01E0);
  f3 = ((crcval>>3) & 0x1FE0) ^ ((crcval<<4) & 0xF000);
  return f1^f2^f3;
}

void
generate_error_table(u8 *bptab, int nbytes)
{
  u16 crc;
  int i, j, k;

  /* Marker for no error */
  bptab[0] = 0xFE;

  /* Marker for >1 error */
  for (i = 1; i < 65536; i++ )
    bptab[i] = 0xFF;

  /* Mark all single bit error cases. */
  printf("Error syndrome table:\n");
  for (i = 0; i < nbytes; i++) {
    putchar(' ');
```

```
            for (j = 0; j < 8; j++) {
               crc = 0;
               for (k = 0; k < i; k++)
                    crc = crc16_t8(crc);
               crc = crc16_t8(crc ^ (0x8000>>j));
               for (k++; k < nbytes; k++)
                    crc = crc16_t8(crc);
               bptab[crc] = (i * 8) + j;
               printf(" %04X",crc);
            }
            putchar('\n');
      }
}

int
main(int argc, char **argv)
{
      u8 buffer[8] = {
         0x01,0x55,0x02,0xaa,
         0x99,0x72,0x18,0x56
      };
      u16 crc;
      int i;

      generate_error_table(sdl_error_position,8);

      /* Run sample message through check routine. */
      crc = 0;
      for (i = 0; i < 8; i++)
         crc = crc16_t8(crc ^ (buffer[i]<<8));

      /* Output is 0000 64 -- no error encountered. */
      printf("\nError test:  CRC %04X, bit position %d\n",
        crc,sdl_message_error_position[crc]);
}
```

7. Security Considerations

The reliability of public SONET/SDH networks depends on well-behaved traffic which does not disrupt the synchronous data recovery mechanisms. This document describes framing and scrambling options that are used to ensure the distribution of transmitted data such that SONET/SDH design assumptions are not likely to be violated.

8. References

[1] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)," RFC 1661, Daydreamer, July 1994.

[2] Simpson, W., Editor, "PPP in HDLC-like Framing," RFC 1662, Daydreamer, July 1994.

[3] Simpson, W., Editor, "PPP over SONET/SDH," RFC 1619, Daydreamer, May 1994.

[4] "American National Standard for Telecommunications - Synchronous Optical Network (SONET) Payload Mappings," ANSI T1.105.02-1993 draft.

[5] ITU-T Recommendation G.707, "Synchronous Digital Hierarchy Bit Rates," June 1992.

[6] Lucent Technologies, "SDL Framer/Frame Inserter," work in progress.

[7] Demers, A., S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," ACM SIGCOMM volume 19 number 4, pp. 1-12, September 1989.

[8] Floyd, S. and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, August 1993.

[9] Simpson, W., "PPP in Ether-like Framing," Daydreamer, work in progress.

[10] Simpson, W., Editor, "PPP LCP Extensions," RFC 1570, Daydreamer, January 1994.

[11] ITU-T Recommendation I.432, "B-ISDN User-Network Interface - Physical Layer Specification," March 1993.

[12] ITU-T Recommendation V.41, "Code-independent error-control system," November 1989.

9. Acknowledgments

PPP over SONET was first proposed by Craig Partridge (BBN), and was last documented by William Simpson as RFC 1619. Much of the material in this document was supplied by Lucent.

10. Working Group and Chair Address

The working group can be contacted via the mailing list (ietf-ppp@merit.edu; send mail to ietf-ppp-request@merit.edu to subscribe), or via the current chair:

Karl Fox
Ascend Communications
655 Metro Place South Suite 370
Dublin OH 43017-3390

Email: karl@ascend.com

11. Legal Issues

11.1. Intellectual Property Notices

11.2. The Authors

Although every effort has been made to secure this information, the authors are unaware of any current or planned patent applications on the fundamental algorithms described in this draft, except within the context of specific and proprietary implementation techniques. Other entities may have claims against this material.

IronBridge Networks has no claim on any of this material.

11.3. Copyright Notice

12. Authors' Addresses

James Carlson
IronBridge Networks
55 Hayden Avenue
Lexington MA 02421-7996
Phone: +1 781 372 8132
Fax: +1 781 372 8090
Email: carlson@ibnets.com

Paul Langner
Lucent Technologies Microelectronics Group
555 Union Boulevard
Allentown PA 18103-1286
Email: plangner@lucent.com

Enrique J. Hernandez-Valencia
Lucent Technologies
101 Crawford Corners Rd.
Holmdel NJ 07733-3030
Email: enrique@lucent.com

James Manchester
Lucent Technologies
101 Crawford Corners Rd.
Holmdel NJ 07733-3030
Email: sterling@hotair.hobl.lucent.com

Appendix K: SDL Packet over Fiber IETF Draft

K.1 Draft

```
PPP Working Group                                  James Carlson
Internet Draft                                IronBridge Networks
Expires December 1999               Enrique J. Hernandez-Valencia
                             Lucent Technologies Bell Laboratories
                                                      Nevin Jones
                        Lucent Technologies Microelectronics Group
                                                     Paul Langner
                        Lucent Technologies Microelectronics Group
                                                        June 1999
```

PPP over Simple Data Link (SDL)
using raw lightwave channels with ATM-like framing
<draft-ietf-pppext-sdl-pol-00.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

To view the list Internet-Draft Shadow Directories, see http://www.ietf.org/shadow.html.

This document is the product of the Point-to-Point Protocol Extensions Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the ietf-ppp@merit.edu mailing list.

Distribution of this memo is unlimited.

Carlson, et al.   expires December 1999   [Page 1]

Abstract

The Point-to-Point Protocol (PPP) in RFC-1661 [1] provides a standard method for transporting multi-protocol datagrams over point-to-point links, and RFCs 1662 [2] and 1619 [3] provide a means to carry PPP over Synchronous Optical Network (SONET) [5] and Synchronous Digital Hierarchy (SDH) [6] circuits. PPP over Simple Data Link (SDL) using SONET/SDH with ATM-like framing (PPPEXT WG work in Progress) extended these standards to include a new encapsulation for PPP called Simple Data Link (SDL) [8]. This document extends the use of SDL over raw lightwave channels without an intervening SONET/SDH layer, which are also referred to as "dark fiber" or Packet-over-Lightwave" (POL) links.

This document is the product of the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the ietf-ppp@merit.edu mailing list.

Applicability

This specification is intended for those implementations which desire to use PPP encapsulation over high speed point-to-point circuits with the so-called "dark fiber" or raw lightwave channels. This enhanced framing mechanisms for PPP encapsulation method has very low overhead, good hardware scaling properties and is resilient to payload expansion. It is anticipated that significantly higher throughput can be attained with SDL when compared to other transport and encapsulation mechanisms for high-speed packet data over lightwave channels, and at a significantly lower cost for line termination equipment.

SDL is defined over other media types and for other data link protocols, but this specification covers only the use of PPP over SDL raw lightwave channels. Systems requiring typical public network functions such as transmission quality assessment, protection switching/restoration, and OAM&P at the transmission level will may require either an additional transmission layer (e.g., SONET/SDH or OTN [7]) with or equivalent OAM&P functionallity not defined in this document.

The use of SDL requires the presentation of packet length information in the SDL header. Thus, hardware implementing SDL must have access to the packet length when generating the header, and where a router's input link does not readily have this information (that is, for non-SDL input links), the router may be required to buffer the entire packet before transmission. "Worm-hole" routing is thus at least problematic with SDL, unless the input links are also SDL. This, however, does not appear to be a great disadvantage on modern routers

due to the general requirement of length information in other parts
of the system, notably in queuing and congestion control strategies
such as Weighted Fair Queuing [12] and Random Early Detection [13].

Table of Contents

1. Introduction

The term packet-over-lightwave (POL) has been used to refer to the capability of transmitting packet data directly over a raw lightwave channel, also referred to as "dark fiber", without an intervening SONET/SDH or optical transport network (OTN) layer. POL solutions are attractive in data networking scenarios were neither multi-segment/multi-path transport nor the OAM&P capabilities of optical transport networking or SONET/SDH is required. SDL on POL does not rely on SONET/SDH or OTN overheads to enable networking features such as transmission quality assessment, protection switching/restoration, and OAM&P. Performance assessment, switching and OAM&P capabilities for SDL on POL are not defined in this document

This document describes a method to enable the use of SDL framing for PPP over such raw lightwave channels and describes the framing and encapsulation requirements for PPP.. The protocol stack is illustrated in Figure 1. While bit-synchronous HDLC-like framing has a worst-case octet overhead of 20% for some specific data patterns, SDL uses no payload encoding, and hence, has zero payload overhead.

```
+--------------------------+
|                          |
| Higher-layer Protocol    |
|                          |
+--------------------------+
|                          |
|          PPP             |
|                          |
+--------------------------+
|                          |
|          SDL             |
|                          |
+--------------------------+
|                          |
| Raw Lightwave Channel    |
|                          |
+--------------------------+
```

Figure 1: Protocol stack for PPP over SDL over a raw lightwave channel.

2. Physical Layer Requirements

The transport mode for SDL on POL is packet-oriented. The raw lightwave links are intrinsically bit-synchronous even though PPP treats the lower transport layer as a full-duplex octet-oriented synchronous interface. No provision is made to support sending or receiving bare octets over lightwaves (as is the case with SONET/SDH).

2.1. Payload Types

Only bit-synchronous payloads at STS-1 and higher line rates are currently considered in this document. Operations at lower bit rates is feasible but not considered at present. Mappings of plesiochronous payloads, such as T1 and T3, on to SDL are not considered in this document.

2.2. Control Signals

A prior-arrangement method is required to enable SDL framing for POL. No LCP-negotiated method is currently proposed. LCP may be used to negotiated other PPP-related parameters (see sections 2.4 and 6).

2.3. Synchronization Modes

Unlike non-SDL O-S encapsulations, SDL provides a positive indication that it has achieved synchronization with the peer. An SDL PPP implementation MUST provide a means to temporarily suspend PPP data transmission (both user data and negotiation traffic) if synchronization loss is detected. An SDL PPP implementation SHOULD also provide a configurable timer that is started when SDL is initialized and restarted on the loss of synchronization, and is terminated when link synchronization is achieved. If this timer expires, implementation-dependent action should be taken to report the hardware failure.

2.4. Framing

PPP over SDL over raw lightwave channels uses the same data link frame format as for PPP over SDL over SONET/SDH [4]. When SDL framing for PPP is employed, the SDL "Datagram Offset" is fixed at 4, and the "A" and "B" messages are not used. Additional information on these optional features of SDL can be found in Lucent's SDL specification [8].

Fixing the Datagram Offset to 4 allows a PPP MRU/MTU of 65536 using SDL.

SDL framing is in general accomplished by the use of a four octet header on the packet. This fixed-length header allows the use of a simple framer to detect synchronization as described in section 2.6. For use with PPP, this header precedes each raw PPP packet as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Packet Length         |          Header CRC            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  PPP packet (beginning with address and control fields)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            .....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Packet CRC                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The four octet length header is DC balanced by exclusive-OR (also known as "modulo 2 addition") with the hex value B6AB31E0. This is the maximum transition, minimum sidelobe, Barker-like sequence of length 32. No other scrambling is done on the header itself.

Packet Length is an unsigned 16 bit number in network byte order. Unlike the standard PPP FCS, the Header CRC is a CRC-16 generated with initial value zero and transmitted in standard network byte order. The PPP packet is scrambled, and begins with the standard address and control fields, and may be any integral octet length (i.e., it is not padded unless the Self Describing Padding option is used). The Packet CRC is also scrambled, and has a mode-dependent length (described below), and is located only on an octet boundary; no alignment of this field may be assumed.

When the Packet Length value is 4 or greater, the distance in octets between one message header and the next in SDL is the sum of Packet Length field, Datagram Offset value, and the fixed size of the Packet CRC field. The Datagram Offset is a configurable SDL parameter, which is set to the fixed value 4 for PPP. When the Packet Length is 0, the distance to the next header is 4 octets. This is the idle fill header. When the Packet Length is 1 to 3, the distance to the next header is 12 octets. These headers are used for special SDL messages used only with optional scrambling and management modes. See section 5 for details of the messages.

General SDL, like PPP, allows the use of no CRC, ITU-T CRC-16, or

ITU-T CRC-32 for the packet data. However, because the Packet Length field does not include the CRC length, synchronization cannot be maintained if the CRC type is changed per RFC 1570, because frame-to-frame distance is, as described above, calculated including the CRC length. Although synchronization can be regained by readjusting the receiver's frame-to-frame distance after a CRC negotiation, this PPP over SDL specification fixes the CRC type to CRC-32 (four octets), and all SDL implementations MUST reject any LCP FCS Alternatives Option [9] requested by the peer when in SDL mode.

PPP over SDL implementations MAY allow a configuration option to set different CRC types for use by prior arrangement. Any such configurable option MUST default to CRC-32, and MUST NOT be include LCP negotiation of FCS Alternatives.

With the SDL Datagram Offset set to 4, the value placed in the Packet Length field is exactly the length in octets of the PPP frame itself, including the address and control fields but not including the FCS field.

Because Packet Lengths below 4 are reserved, the Packet Length MUST be 4 or greater for any legal PPP packet. PPP packets with fewer octets, which are not possible without address/control or protocol field compression, MUST be padded to length 4 for SDL.

Inter-packet time fill is accomplished by sending the four octet length header with the Packet Length set to zero. No provision is made for intra-packet time fill.

By default, an independent, self-synchronous x^43+1 scrambler is used on the data portion of the message including the 32 bit CRC. This is done in exactly the same manner as with the ATM x^43+1 scrambler on a SONET/SDH link. The scrambler is not clocked when SDL header bits are transmitted. Thus, the data scrambling can be implemented in an entirely independent manner from the SDL framing.

Optionally, by prior arrangement, SDL links MAY use a set-reset scrambler as described in section 2.9. If this option is provided, it MUST be configurable by the administrator, and the option MUST default to the self-synchronous scrambler.

Once the link enters SYNCH state, the SDL header single bit error correction logic is enabled (see section 2.9). Any unrecoverable header CRC error returns the link to HUNT state, disables PPP transmission, and disables the error correction logic.

2.5. Synchronization Procedure

The link synchronization procedure is similar to the I.432 section 4.5.1.1 ATM HEC delineation procedure [10], except that the SDL messages are variable length. The machine starts in HUNT state until a four octet sequence in the data stream with a valid CRC-16 is found. (Note that the CRC-16 single-bit error correction technique described in section 2.9 is not employed until the machine is in in SYNCH state. The header must have no bit errors in order to leave HUNT state.) Such a valid sequence is a candidate SDL header. On finding the valid sequence, the machine enters PRESYNCH state. Any one invalid SDL header in PRESYNCH state returns the link to HUNT state.

If a second valid SDL header is seen after entering PRESYNCH state, then the link enters SYNCH state and PPP transmission is enabled. If an invalid SDL header is detected, then the link is returned to HUNT state without enabling PPP transmission.

2.6. Scrambler Operation

The transmit and receive scramblers are shift registers with 43 stages that MAY be initialized to all-ones when the link is initialized. Synchronization is maintained by the data itself.

```
        Transmit                      Receive

 DATA-STREAM (FROM PPP)          IN (FROM SDL FRAMER)
 |                    |
 v                    |
 XOR<-----------------+    +->D0-+->D1-> ... ->D41->D42-+
 |                 |  |                   |
 +->D0-+->D1-> ... ->D41->D42-+   XOR<--------------------+
 |                    |
 v                    v
 OUT (TO SDL FRAMER)             DATA-STREAM (TO PPP)
```

Each XOR is an exclusive-or gate; also known as a modulo-2 adder. Each Dn block is a D-type flip-flop clocked on the appropriate data clock.

The scrambler is clocked once after transmission or reception of each bit of payload and before the next bit is applied as input. Bits within an octet are, per SONET/SDH standard practice, transmitted and received MSB-first.

2.7. CRC Generation

The CRC-16 and CRC-32 generator polynomials used by SDL are the ITU-T standard polynomials [11]. These are:

x^16+x^12+x^5+1

x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1

The SDL Header CRC and the CRC-16 used for each of the three special messages (scrambler state, message A, and message B; see section 5) are all generated using an initial remainder value of 0000 hex.

The optional CRC-16 on the payload data (this mode is not used with PPP over SDL except by prior arrangement) uses the standard initial remainder value of FFFF hex for calculation and the bits are complemented before transmission. The final CRC remainder, however, is transmitted in network byte order, unlike the regular PPP FCS. If the CRC-16 algorithm is run over all of the octets including the appended CRC itself, then the remainder value on intact packets will always be E2F0 hex. Alternatively, an implementation may stop CRC calculation before processing the appended CRC itself, and do a direct comparison.

The standard CRC-32 on the payload data (used for standard PPP over SDL) uses the initial remainder value of FFFFFFFF hex for calculation and the bits are complemented before transmission. The CRC, however, is transmitted in network byte order, most significant bit first, unlike the optional PPP 32 bit FCS, which is transmitted in reverse order. The remainder value on intact packets when the appended CRC value is included in the calculation is 38FB2284.

C code to generate these CRCs is found in Appendix A.

2.8. Error Correction

The error correction technique is based on the use of a Galois number field, as with the ATM HEC correction. In a Galois number field, f(a+b) = f(a) + f(b). Since the CRC-16 used for SDL forms such a field, we can state that CRC(message+error) = CRC(message) + CRC(error). Since the CRC-16 remainder of a properly formed message is always zero, this means that, for the N distinct "error" strings corresponding to a single bit error, there are N distinct CRC(error) values, where N is the number of bits in the message.

A table look-up is thus applied to the CRC-16 residue after calculation over the four octet SDL header to correct bit errors in the header and to detect multiple bit errors. For the optional set-reset scrambler, a table look-up is similarly applied to the CRC-16 residue after calculation over the eight octet scrambler state message to correct bit errors and to detect multiple bit errors. (This second correction is also used for the special SDL A and B messages, which are not used for standard PPP over SDL.)

Note: No error correction is performed for the payload.

Note: This error correction technique is used only when the link has entered SYNCH state. While in HUNT or PRESYNCH state, error correction should not be performed, and only messages with syndrome 0000 are accepted. If the calculated syndrome does not appear in this table, then an unrecoverable error has occurred. Any such error in the SDL header will return the link to HUNT state.

Since the CRC calculation is started with zero, the two tables can be merged. The four octet table is merely the last 32 entries of the eight octet table.

Eight octet (64 bit) single bit error syndrome table (in hex):

```
FD81 F6D0 7B68 3DB4 1EDA 0F6D 8FA6 47D3
ABF9 DDEC 6EF6 377B 93AD C1C6 60E3 B861
D420 6A10 3508 1A84 0D42 06A1 8B40 45A0
22D0 1168 08B4 045A 022D 8906 4483 AA51
DD38 6E9C 374E 1BA7 85C3 CAF1 ED68 76B4
3B5A 1DAD 86C6 4363 A9A1 DCC0 6E60 3730
1B98 0DCC 06E6 0373 89A9 CCC4 6662 3331
9188 48C4 2462 1231 8108 4084 2042 1021
```

Thus, if the syndrome 6EF6 is seen on an eight octet message, then the third bit (hex 20) of the second octet is in error. Similarly, if 48C4 is seen on an eight octet message, then the second bit (hex 40) in the eighth octet is in error. For a four octet message, the same two syndromes would indicate a multiple bit error for 6EF6, and a single bit error in the second bit of the fourth octet for 48C4.

Note that eight octet messages are used only for the optional set-reset scrambling mode, described in section 6.

Corresponding C code to generate this table is found in Appendix B.

3. Performance Analysis

There are five general statistics that are important for framing algorithms. These are:

MTTF	Mean time to frame
MTTS	Mean time to synchronization
PFF	Probability of false frame
PFS	Probability of false synchronization
PLF	Probability of loss of frame

The following sections summarize each of these statistics for SDL. Details and mathematic development can be found in the Lucent SDL documentation [8].

3.1. Mean Time To Frame (MTTF)

This metric measures the amount of time required to establish correct framing in the input data. This may be measured in any convenient units, such as seconds or bytes. For SDL, the relevant measurement is in packets, since fragments of packets are not useful.

In order to calculate MTTF, we must first determine how often the frame detection state machine is "unavailable" because it failed to detect the next incoming SDL frame within the user data.

Since the probability of a false header detection using CRC-16 in random data is 2^-16 and this rate is large compared to the allowable packet size, it is worthwhile to run multiple parallel frame-detection state machines. Each machine starts with a different candidate framing point in order to reduce the probability of falsely detecting user data as a valid frame header.

The results for this calculation for 8KB packets and 384 byte packets are:

Number of Framers	Unavailability 8KB packets	Unavailability 384 byte pkts
1	3.68E-1	1.55E-2
2	1.04E-1	1.60E-4
3	2.32E-2	1.25E-6
4	4.35E-3	7.75E-9

Using these values, MTTF can be calculated as a function of the Bit Error Rate (BER). These plots show a characteristically flat region for all BERs up to a knee, beyond which the begins to rise sharply. In all cases, this knee point has been found to occur at a BER of approximately 1E-4, which is several orders of magnitude above that observed on existing SONET/SDH links. The flat rate values are summarized as:

Number of Flat region Flat region

Framers	8KB packets	384 bytes
1	2.08	1.52
2	1.62	1.50
3	1.52	1.50
4	1.50	1.50

Thus, for common packet sizes in an implementation with two parallel framers using links with a BER of 1E-4 or better, the MTTF is approximately 1.5 packets. This is also the optimal time, since it represents initiating framing at an average point half-way into one packet, and achieving good framing after seeing exactly one correctly framed packet.

3.2. Mean Time To Synchronization (MTTS)

The MTTS for standard SDL with a self-synchronous scrambler is the same as the MTTF, or 1.5 packets.

The MTTS for SDL using the optional set-reset scrambler is one half of the scrambling state transmission interval (in packets) plus the MTTF. For insertion at the default rate of one per eight packets, the MTTS is 5.5 packets.

(The probability of receiving a bad scrambling state transmission should also be included in this calculation. The probability of random corruption of this short message is shown in the SDL document [8] to be small enough that it can be neglected for this calculation.)

3.3. Probability of False Frame (PFF)

The PFF is 232.8E-12 (2^-32), since false framing requires two consecutive headers with falsely correct CRC-16.

3.4. Probability of False Synchronization (PFS)

The PFS for the standard self-synchronous scrambler is the same as the PFF, or 232.8E-21 (2^-32).

The PFS for the set-reset scrambler is 54.21E-21 (2^-64), and is calculated as the PFF above multiplied by the probability of a falsely detected scrambler state message, which itself contains two independent CRC-16 calculations.

3.5. Probability of Loss of Frame (PLF)

The PLF is a function of the BER, and for SDL is approximately the square of the BER multiplied by 500, which is the probability of two or more bit errors occurring within the 32 bit SDL header. Thus, at a BER of 1E-5, the PLF is 5E-8.

4. The Special Messages

When the SDL Packet Length field has any value between 0000 and 0003, the message following the header has a special, pre-defined length. The 0 value is a time-fill on an idle link, and no other data follows. The next octet on the link is the first octet of the next SDL header.

The values 1 through 3 are defined in the following subsections. These special messages each consist of a six octet data portion followed by another CRC-16 over that data portion, as with the SDL header, and this CRC is used for single bit error correction.

4.1. Scrambler State

The special value of 1 for Packet Length is reserved to transfer the scrambler state from the transmitter to the receiver for the optional set-reset scrambler. In this case, the SDL header is followed by six octets (48 bits) of scrambler state. Neither the scrambler state nor the CRC are scrambled.

4.2. A/B Message

The special values of 2 and 3 for Packet Length are reserved for "A" and "B" messages, which are also six octets in length followed by two octets of CRC-16. Each of these eight octets are scrambled. No use for these messages with PPP SDL is defined. These messages are reserved for use by link maintenance protocols, in a manner analogous to ATM's OAM cells.

5. The Set-Reset Scrambler Option

Standard PPP over SDL uses a self-synchronous scrambler. SDL implementations MAY also employ a set-reset scrambler to avoid some of the possible inherent problems with self-synchronous scramblers.

5.1. The Killer Packet Problem

Scrambling in general solves two problems. First, most line interfaces (e.g., SONET/SDH) require a minimum density of bit transitions in order to maintain hardware clock recovery. Since data streams frequently contain long runs of all zeros or all ones, scrambling the bits using a pseudo-random number sequence breaks up these patters. Second, all link-layer synchronization mechanisms rely on detecting long-range patterns in the received data to detect framing.

Self-synchronous scramblers are an easy way to partially avoid these problems. One problem that is inherent with self-synchronous, however, is that long user packets from malicious sites can make use of the known properties of these scramblers to inject either long strings of zeros or other synchronization-destroying patterns into the link.

Such carefully constructed packets are called "killer packets."

5.2. SDL Set-Reset Scrambler

An alternative to the self-synchronous scrambler is the externally synchronized or "set-reset" scrambler. This is a free-running scrambler that is not affected by the patterns in the user data, and therefore minimizes the possibility that a malicious user could present data to the network that mimics an undesirable data pattern.

The option set-reset scrambler defined for SDL is an x^48+x^28+x^27+x+1 independent scrambler initialized to all ones when the link enters PRESYNCH state and reinitialized if the value ever becomes all zero bits. As with the self-synchronous scrambler, all octets in the PPP packet data following the SDL header through the final packet CRC are scrambled.

5.3. SDL Scrambler Synchronization

As described in the previous section, the special value of 1 for Packet Length is reserved to transfer the scrambler state from the transmitter to the receiver. In this case, the SDL header is followed by six octets (48 bits) of scrambler state plus two octets of CRC-16 over the scrambler state. None of these eight octets are scrambled.

SDL synchronization consists of two components, link and scrambler synchronization. Both must be completed before PPP data flows on the link.

If a valid SDL header is seen in PRESYNCH state, then the link enters SYNCH state, and the scrambler synchronization sequence is started. If an invalid SDL header is detected, then the link is returned to HUNT state, and PPP transmission is suspended.

When scrambler synchronization is started, a scrambler state message is sent (Packet Length set to 1 and six octets of scrambler state in network byte order follow the SDL header). This message is sent once. At this point, PPP transmission is enabled.

Scrambler state messages are periodically transmitted to keep the peers in synchronization. A period of once per eight transmitted packets is suggested, and it SHOULD be configurable. Excessive packet CRC errors detected indicates an extended loss of synchronization and should trigger link resynchronization.

On reception of a scrambler state message, an SDL implementation MUST compare the received 48 bits of state with the receiver's scrambler state. If any of these bits differ, then a synchronization slip error is declared. After such an error, the next valid scrambler state message received MUST be loaded into the receiver's scrambler, and the error condition is then cleared.

5.4. SDL Scrambler Operation

The transmit and receive scramblers are shift registers with 48 stages that are initialized to all-ones when the link is initialized. Each is refilled with all one bits if the value in the shift register ever becomes all zeros. This scrambler is not reset at the beginning of each frame, as is the SONET/SDH X^7+X^6+1 scrambler, nor is it modified by the transmitted data, as is the ATM self-synchronous scrambler. Instead it is kept in synchronization using special SDL messages.

```
+----XOR<-------------XOR<--XOR<----------------+
|    ^              ^   ^             |
|    |              |   |             |
+->D0-+->D1-> ... ->D26--+->D27-+->D28-> ... ->D47-+
|
v
OUT
```

Each XOR is an exclusive-or gate; also known as a modulo-2 adder. Each Dn block is a D-type flip-flop clocked on the appropriate data clock.

The scrambler is clocked once after transmission of each bit of SDL data, whether or not the transmitted bit is scrambled. When scrambling is enabled for a given octet, the OUT bit is exclusive-ored with the raw data bit to produce the transmitted bit. Bits within an octet are transmitted MSB-first.

Reception of scrambled data is identical to transmission. Each received bit is exclusive-ored with the output of the separate receive data scrambler.

To generate a scrambler state message, the contents of D47 through D0 are snapshot at the point where the first scrambler state bit is sent. D47 is transmitted as the first bit of the output. The first octet transmitted contains D47 through D40, the second octet D39 through D32, and the sixth octet D7 through D0.

The receiver of a scrambler state message MUST first run the CRC-16 check and correct algorithm over this message. If the CRC-16 message check detects multiple bit errors, then the message is dropped and is not processed further.

Otherwise, it then should compare the contents of the entire receive scrambler state D47:D0 with the corrected message. (By pipelining the receiver with multiple clock stages between SDL Header error-correction block and the descrambling block, the receive descrambler will be on the correct clock boundary when the message arrives at the descrambler. This means that the decoded scrambler state can be treated as immediately available at the beginning of the D47 clock cycle into the receive scrambler.)

If any of the received scrambler state bits is different from the corresponding shift register bit, then a soft error flag is set. If the flag was already set when this occurs, then a synchronization slip error is declared. This error SHOULD be counted and reported through implementation-defined network management procedures. When the receiver has this soft error flag set, any scrambler state message that passes the CRC-16 message check without multiple bit errors is clocked directly into the receiver's state register after the comparison is done, and the soft error flag is then cleared. Otherwise, while uncorrectable scrambler state messages are received, the soft error flag state is maintained.

(The intent of this mechanism is to reduce the likelihood that a falsely corrected scrambler state message with multiple bit errors can corrupt the running scrambler state.)

6. Configuration Details

The following PPP Configuration Options are recommended:

Magic Number
No Address and Control Field Compression
No Protocol Field Compression
No FCS alternatives (32-bit FCS default)

This configuration means that standard PPP over SDL on POL generally presents a 32-bit aligned datagram to the network layer. With the address, control, and protocol field intact, the PPP overhead on each packet is four octets. If the SDL framer presents the SDL packet header to the PPP input handling in order to communicate the packet length , this header is also four octets, and word-alignment is preserved.

Since SDL does take the place of HDLC as a transport for PPP, it is at least tempting to remove the HDLC-derived overhead. This is not done for standard PPP over SDL in order to preserve the message alignment and the future possibility of Frame Relay internetworking.

By prior external arrangement, any two SDL implementations MAY omit the address and control fields or implement protocol field compression. Such use is not standardized and MUST NOT be the default on any SDL implementation.

Appendix A: CRC Generation

The following unoptimized code generates proper CRC-16 and CRC-32 values for SDL messages. Note that the polynomial bits are numbered in big-endian order for SDL CRCs; bit 0 is the MSB.

```
 typedef unsigned char u8;
 typedef unsigned short u16;
 typedef unsigned long u32;

 #define POLY16  0x1021
 #define POLY32  0x04C11DB7

u16
crc16(u16 crcval, u8 cval)
{
   int i;

   crcval ^= cval << 8;
   for (i = 8; i--; )
      crcval = crcval & 0x8000 ? (crcval << 1) ^ POLY16 :
         crcval << 1;
   return crcval;
}

u32
crc32(u32 crcval, u8 cval)
{
   int i;

   crcval ^= cval << 24;
   for (i = 8; i--; )
      crcval = crcval & 0x80000000 ? (crcval << 1) ^ POLY32 :
         crcval << 1;
   return crcval;
}

u16
crc16_special(u8 *buffer, int len)
{
   u16 crc;

   crc = 0;
   while (--len >= 0)
      crc = crc16(crc,*buffer++);
   return crc;
}
```

```
u16
crc16_payload(u8 *buffer, int len)
{
   u16 crc;

   crc = 0xFFFF;
   while (--len >= 0)
      crc = crc16(crc,*buffer++);
   return crc ^ 0xFFFF;
}

u32
crc32_payload(u8 *buffer, int len)
{
   u32 crc;

   crc = 0xFFFFFFFFul;
   while (--len >= 0)
      crc = crc32(crc,*buffer++);
   return crc ^ 0xFFFFFFFFul;
}

void
make_sdl_header(int packet_length, u8 *buffer)
{
   u16 crc;

   buffer[0] = (packet_length >> 8) & 0xFF;
   buffer[1] = packet_length & 0xFF;
   crc = crc16_special(buffer,2);
   buffer[0] ^= 0xB6;
   buffer[1] ^= 0xAB;
   buffer[2] = ((crc >> 8) & 0xFF) ^ 0x31;
   buffer[3] = (crc & 0xFF) ^ 0xE0;
}
```

Appendix B: Error Correction Tables

To generate the error correction table, the following implementation may be used. It creates a table called sdl_error_position, which is indexed on CRC residue value. The tables can be used to determine if no error exists (table entry is equal to FE hex), one correctable error exists (table entry is zero-based index to errored bit with MSB of first octet being 0), or more than one error exists, and error is uncorrectable (table entry is FF hex). To use for eight octet messages, the bit index from this table is used directly. To use for four octet messages, the index is treated as an unrecoverable error if it is below 32, and as bit index plus 32 if it is above 32.

The program also prints out the error syndrome table shown in section 2.9. This may be used as part of a "switch" statement in a hardware implementation.

```
u8 sdl_error_position[65536];

/* Calculate new CRC from old^(byte<<8) */
u16
crc16_t8(u16 crcval)
{
  u16 f1,f2,f3;

  f1 = (crcval>>8) | (crcval<<8);
  f2 = (crcval>>12) | (crcval&0xF000) | ((crcval>>7)&0x01E0);
  f3 = ((crcval>>3) & 0x1FE0) ^ ((crcval<<4) & 0xF000);
  return f1^f2^f3;
}

void
generate_error_table(u8 *bptab, int nbytes)
{
  u16 crc;
  int i, j, k;

  /* Marker for no error */
  bptab[0] = 0xFE;

  /* Marker for >1 error */
  for (i = 1; i < 65536; i++ )
    bptab[i] = 0xFF;

  /* Mark all single bit error cases. */
  printf("Error syndrome table:\n");
  for (i = 0; i < nbytes; i++) {
    putchar(' ');
```

```
          for (j = 0; j < 8; j++) {
            crc = 0;
            for (k = 0; k < i; k++)
                crc = crc16_t8(crc);
            crc = crc16_t8(crc ^ (0x8000>>j));
            for (k++; k < nbytes; k++)
                crc = crc16_t8(crc);
            bptab[crc] = (i * 8) + j;
            printf(" %04X",crc);
          }
          putchar('\n');
      }
}

int
main(int argc, char **argv)
{
    u8 buffer[8] = {
      0x01,0x55,0x02,0xaa,
      0x99,0x72,0x18,0x56
    };
    u16 crc;
    int i;

    generate_error_table(sdl_error_position,8);

    /* Run sample message through check routine. */
    crc = 0;
    for (i = 0; i < 8; i++)
      crc = crc16_t8(crc ^ (buffer[i]<<8));

    /* Output is 0000 64 -- no error encountered. */
    printf("\nError test:  CRC %04X, bit position %d\n",
     crc,sdl_message_error_position[crc]);
}
```

7. Security Considerations

The reliability of communication networks places special requirements in the handling of data payloads as appropriate to the specific line encoding schemes. This document describes framing and scrambling options for SDL over raw lightwave channels that enable the use of current typical design (non burst mode) optical transceiver and timing subsystem. In particular, this proposal is compatible with DWDM regenertor networks. No other security concerns have been identified.

8. Intellectual Properties Considerations

Lucent Technologies Inc. may own intellectual property on some of the technologies disclosed in this document. The patent licensing policy of Lucent Technologies Inc. with respect to any patents or patent applications relating to this submission is stated in the March 1, 1999, letter to the IETF from Dr. Roger E. Stricker, Intellectual Property Vice President, Lucent Technologies Inc. This letter is on file in the offices of the IETF Secretariat.

IronBridge Networks has no claim on any of this material.

9. References

[1] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)," RFC 1661, Daydreamer, July 1994.

[2] Simpson, W., Editor, "PPP in HDLC-like Framing," RFC 1662, Daydreamer, July 1994.

[3] Simpson, W., Editor, "PPP over SONET/SDH," RFC 1619, Daydreamer, May 1994.

[4] Carlson, Langner, Hernandez-Valencia, Manchester, "PPP over Simple Data Link (SDL) using SONET/SDH with ATM-like framing," PPPEXT WG work in progress.

[5] "American National Standard for Telecommunications - Synchronous Optical Network (SONET) Payload Mappings," ANSI T1.105.02-1995.

[6] ITU-T Recommendation G.707, "Network Node Interface for the Synchronous Digital Hierarchy (SDH)," March 1996.

[7] ITU-T Recommendation G.872, "Architecture of Optical

Transport Networks." February 1999.

[8] Doshi,B., Dravida, S., Hernandez-Valencia, E., Matragi, W., Qureshi, M., Anderson, J., Manchester, J.,"A Simple Data Link Protocol for High Speed Packet Networks", Bell Labs Technical Journal, pp. 85-104, Vol.4 No.1, January-March 1999.

[9] Simpson, W., Editor, "PPP LCP Extensions," RFC 1570, Daydreamer, January 1994.

[10] ITU-T Recommendation I.432.1, "B-ISDN User-Network Interface - Physical Layer Specification: General Characteristics," February 1999.

[11] ITU-T Recommendation V.41, "Code-independent error-control system," November 1989.

[12] Demers, A., S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," ACM SIGCOMM volume 19 number 4, pp. 1-12, September 1989.

[13] Floyd, S. and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, August 1993.

10. Acknowledgments

The authors recognize Jon Anderson, Bharat Doshi, Subra Dravida and James Manchester from Lucent Technologies for their various contributions to this work.

11. Working Group and Chair Address

The working group can be contacted via the mailing list (ietf-ppp@merit.edu; send mail to ietf-ppp-request@merit.edu to subscribe), or via the current chair:

Karl Fox
Extant Networks

Email: karl@extant.com

12. Authors' Addresses

James Carlson

IronBridge Networks
55 Hayden Avenue
Lexington MA 02421-7996
Phone: +1 781 372 8132
Fax: +1 781 372 8090
Email: carlson@ibnets.com

Enrique J. Hernandez-Valencia
Lucent Technologies Bell Laboratories
101 Crawford Corners Rd.
Holmdel NJ 07733-3030
Email: enrique@lucent.com

Nevin Jones
Lucent Technologies Microelectronics Group
555 Union Boulevard
Allentown PA 18103-1286
Email: nrjones@lucent.com

Paul Langner
Lucent Technologies Microelectronics Group
555 Union Boulevard
Allentown PA 18103-1286
Email: plangner@lucent.com

References:

[1] R. Zakon, "*Hobbes' Internet Timeline*", RFC2235, Internet Engineering Task Force, November 1997.

[2] John P. Ryan, "*WDM: North American Deployment Trends*", IEEE Communications Magazine, February 1998.

[3] "*What's the Most Popular Application?*", http://www.nlanr.net/NA/Learn/popular.html, National Laboratory for Applied Network Research (NLANR), April 1998.

[4] J. Postel, "*User Datagram Protocol*", RFC768, Internet Engineering Task Force, August 1980.

[5] J. Postel, "*Internet Protocol*", RFC791, Internet Engineering Task Force, September 1981.

[6] J. Postel, "*Transmission Control Protocol*", RFC793, Internet Engineering Task Force, September 1981.

[7] "*Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria*", GR-253-CORE (A Module of TSGR, FR-440) Issue 2, Revision 1, Bellcore, December 1997.

[8] "*Network Node Interfaces for the Synchronous Digital Hierarchy (SDH)*", ITU-T Recommendation G.707, International Telecommunication Union, March 1996.

[9] "*B-ISDN User-Network Interface - Physical Layer Specification*", ITU-T Recommendation I.432, International Telecommunication Union, 1993.

[10] "*B-ISDN ATM Adaptation Layer (AAL) Specification*", ITU-T Recommendation I.363, International Telecommunication Union, 1993.

[11] "*ATM User-Network Interface (UNI) Specification Version 3.1*", ATM Forum, 1994.

[12] "*B-ISDN ATM Adaptation Layer specification: Type 5 AAL*", ITU-T Recommendation I.363.5, International Telecommunication Union, 1993.

[13] Juha Heinanen, "*Multiprotocol Encapsulation over ATM Adaptation Layer 5*", RFC1483, Internet Engineering Task Force, July 1993.

[14] R.Cole, D. Shur, C. Villamizar, "IP over ATM: A Framework Document", RFC1932, Internet Engineering Task Force, April 1996.

[15] W. Simpson, "*The Point-to-Point Protocol*", RFC1661, Internet Engineering Task Force, July 1994.

[16] W. Simpson, "*PPP Over SONET/SDH*", RFC1619, Internet Engineering Task Force, May 1994.

[17] W. Simpson, "*PPP in HDLC-like Framing*", RFC1662, Internet Engineering Task Force, July 1994.

[18] J. Manchester, J. Anderson, B. Dhoshi, S. Dravida, "*IP Over Optical WDM Transport Networking*", NFOEC '98, Thursday September 17, 1998, Session 30.

[19] Erik Nordmark, "*WAN Packet Size Distribution*", http://www.nlanr.net/NA/Learn/packetsizes.html, National Laboratory for Applied Network Research (NLANR), July 1997.

[20] T. Li, Y. Rekhter "*A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)*", RFC2430, Internet Engineering Task Force, October, 1998.

[21] E. Rosen, Y. Rekhter, "*BGP/MPLS VPNs*", RFC1662, Internet Engineering Task Force, March, 1999.

[22] "*UTOPIA Level 2 Version 1.0*", af-phy-0039.000, ATM Forum, 1995.

[23] Bill Waggener, "*Pulse Code Modulation Techniques*", New York: Van Nostrand Reinhold, 1995. Chapter 7: "Format Synchronization", pp. 313 - 353.

[24] Abraham Wald, "*Sequential Analysis*", New York: John Wiley, 1947.

[25] S.J. Mason, "*Feedback Theory - Further Properties of Signal Flow Graphs*", Proceedings of the IRE, July, 1956.

[26] "*T7630 Dual T1/E1 Short-Haul Terminator (Terminator-II) Preliminary Data Sheet*", Lucent Microelectronics, October 1997.

[27] J.E. Savage, "*Some Simple Self-Synchronizing Digital Data Scramblers*", Bell System Technical Journal, February 1967, pp. 449 - 487.

[28] John G. Proakis, "*Digital Communications*", New York: McGraw-Hill, 1995. Section 13-2-4: "Generation of PN Sequences", pp. 724 - 729.

[29] S.W. Golomb, "*Shift Register Sequences (Revised Edition)*", Aegean Park Press, 1982.

[30] I.J. Fair, V.K. Bhargava, Q. Wang, "*On the Power Spectral Density of Self-Synchronizing Scrambled Sequences*", IEEE Transactions on Information Theory, July 1998, pp. 1687 - 1693.

[31] "*Random Number Generators*", http://csep1.phy.ornl.gov/rn/rn.html, Computational Sciences Educational Project, 1995.

[32] David A. Fisher, Simon D. Brueckheimer, "*Sequence Synchronization*", U.S. Patent #5,237,593, August 17, 1993.

[33] David A. Fisher, Simon D. Brueckheimer, "*Sequence Synchronization*", U.S. Patent #5,321,754, June 14, 1994.

[34] Stephen B. Wicker, "*Error Control Systems for Digital Communication and Storage*", New Jersey: Prentice-Hall, 1995, Page 217-224.

[35] Stephen B. Wicker, "*Error Control Systems for Digital Communication and Storage*", New Jersey: Prentice-Hall, 1995, pp. 75 - 76.

[36] Stephen B. Wicker, "*Error Control Systems for Digital Communication and Storage*", New Jersey: Prentice-Hall, 1995, Page 122.

[37] Joseph J. DiStephano, III, Allen R. Stubberud, Ivan J. Williams, "*Theory and Problems of Feedback and Control Systems*", Schaum's Outline Series, New York: McGraw-Hill, 1967, Page 136-161.

[38] Charles M. Close, Dean K. Frederick, "*Modelling and Analysis of Dynamic Systems*", Boston: Houghton Mifflin Company, 1978, Page 572.

[39] Richard E. Blahut, "*Digital Transmission of Information*", Boston: Houghton Mifflin Company, 1978, pp. 429 - 432.

[40] Gordon Moore, Intel, 1965.

[41] D. Borman, "*TCP and UDP Over IPv6 Jumbograms*", RFC2147, Internet Engineering Task Force, May 1997.

[42] I. Miller, J.E. Freund, "*Probability and Statistics for Engineers*", New Jersey: Prentice Hall, 1977, pp. 60 - 61.

[43] Morris Kline, "*Mathematical Thought From Ancient To Modern Times: Volume 2*", New York: Oxford University Press, 1990, pp. 450 - 451.

[44] Murray R. Spiegel, "*Mathematical Handbook*", New York: McGraw-Hill, 1968, pp. 107 - 108.

[45] Carl-Erik Fröberg, "*Introduction to Numerical Analysis*", Massachusetts: Addison-Wesley, 1969, pp. 224 - 225.

[46] "*Digital Line System based on the Synchronous Digital Hierarchy for use one Optical Fiber Cables*", ITU-T Recommendation G.958, International Telecommunication Union.

[47] Stephen B. Wicker, "*Error Control Systems for Digital Communication and Storage*", New Jersey: Prentice-Hall, 1995, pp. 29 - 30.

[48] S. Lang, "*Undergraduate Algebra*", New York: Springer-Verlag, 1990, pp. 251 - 253

[49] W.A Adkins, S.H. Weintraub, "*Algebra: An Approach via Module Theory*", pp. 92 - 98.

[50] Stephen B. Wicker, "*Error Control Systems for Digital Communication and Storage*", New Jersey: Prentice-Hall, 1995, Page 112-116.

[51] "*UTOPIA Level 2, Version 1.0*", af-phy-0039.000, ATM Forum, June, 1995.