A MESSAGE PRIORITY LOCAL AREA NETWORK FOR SUBSTATION PROTECTION AND CONTROL

JOSE CHOI, B.Eng.

A MESSAGE PRIORITY LOCAL AREA NETWORK FOR SUBSTATION PROTECTION AND CONTROL

By

©Jose Choi, B.Eng.

A thesis submitted to the school of Graduate
Studies in partial fulfilment of the
requirements for the degree of
Master of Engineering

Faculty of Engineering and Applied Science
Memorial University of Newfoundland
St. John's, Newfoundland
Canada

December 1989

ISBN 0-315-59232-X

ABSTRACT

A specialized microcomputer network is developed to meet the real-time processing requirements of a digitally protected substation. This application has strict timing constraints and requires a fast communication response. The network is based on the principle that direct access to channel is granted exclusively to the ready message with the current highest priority.

The network requirements of the digitally protected substation are defined. The data flow of the substation resembles the message priority in nature. To implement the message priority scheme, a totally distributed network is employed.

A simulation model is developed to predict the performance of the network. Results from the simulation indicate that the network is capable of meeting the timing constraints of the real-time application.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. M. A. Rahman for his guidance, support and patience throughout our association. His assistance was instrumental in defining the research project and his critical review of the text during preparation was an invaluable asset. I also thank Dr. D.W. Close, associate professor of Political Science for his assistance during preparation of the final version of this work.

My sincere thanks are also due to Dr. N. Ekanayake, Dr. B. Jeyasurya and Mr. Kin Lam for their invaluable help during this research. A note of thanks to the staff and graduate students of the faculty of Engineering and Applied Science for their contributions via discussion of topics related to the thesis.

CONTENTS

		Page
ABSTRACT		iii
ACKNOWLEDGEMENTS		iv
CONTENTS		v
LIST OF FIGURES		viii
LIST OF ABBREVIATIONS		ix
1.	INTRODUCTION	1
1.1	REVIEW OF CURRENT RESEARCH	2
1.2	RESEARCH APPROACH	3
1.3	CONTENTS OF THE THESIS	4
2.	DIGITAL PROTECTION AND CONTROL IN POWER SUBSTATION	6
2.1	POWER SYSTEM FAULTS & PROTECTIVE RELAYS	6
2.1.1	THE FUNCTION OF PROTECTIVE RELAYS	8
2.1.2	TYPE OF PROTECTIVE RELAYS	8
2.1.3	THE SHORTCOMINGS OF CONVENTIONAL RELAYS	9
2.2	THE DIGITAL RELAYS	10
2.2.1	THE CONCEPTS OF DIGITAL PROTECTION SCHEME	10
2.2.2	THE ADVANTAGES OF DIGITAL RELAYS	12
2.3	INTEGRATED MICROCOMPUTER BASED SUBSTATION PROTECTION AND CONTROL	13
2.3.1	THE NEEDS FOR TRANSMISSION PROTECTION AND CONTROL	14
2.3.1.1	NEED FOR FAST SPEED RESPONSE AND SYSTEM DEPENDABILITY	14
2.3.1.2	NEED FOR CENTRALIZED CONTROL FUNCTIONS	14
2.3.1.3	LARGE DISTANCES	14
2.3.1.4	WITHSTANDING THE ELECTROMAGNETIC ENVIRONMENT	15
2.3.1.5	NEED FOR EVENT RECORDING	15
2.3.1.6	NEED FOR UNATTENDED OPERATION	15
2.3.1.7	NEED FOR SYSTEM WIDE CONTROL	16
2.3.1.8	NEED FOR PERIODIC MODIFICATION	16
2.3.2	SUBSTATION FUNCTIONAL REQUIREMENTS	16
2.3.2.	PROTECTION FUNCTIONS	17
2.3.2.2	AUTOMATIC CONTROL FUNCTIONS	17
2.3.2.3	MONITORING FUNCTIONS	18
2.3.2.4	RECORDING AND DISPLAYING FUNCTIONS	18
2.3.3	SYSTEM RELIABILITY	19
2.3.4	SUMMARY	19

3.	LOCAL COMPUTER NETWORKING	20
3.1	LOCAL AREA NETWORKS (LANS)	20
3.1.1	TRANSMISSION MEDIA	21
3.1.2	INTERFACE	23
3.1.3	PROTOCOL CONTROL	24
3.1.4	END USERS	24
3.2	TOPOLOGIES	24
3.3	LANS PROTOCOL	26
3.3.1	CARRIER SENSE NETWORK	27
3.3.2	CSMA PROTOCOL	28
3.3.2.1	1-PERSISTENT CSMA	28
3.3.2.2	NON-PERSISTENT CSMA	29
3.3.2.3	P-PERSISTENT CSMA	29
3.3.2.4	CSMA/CD	30
3.3.2.5	COLLISION FREE PROTOCOLS	32
3.3.3	RING NETWORKS	34
3.3.3.1	TOKEN RING	34
3.4	LANS STANDARDS	36
3.5	SUMMARY	37
4.	THE ARCHITECTURE OF THE SUBATATION PROTECTION AND CONTROL SYSTEM	39
4.1	SYSTEM ARCHITECTURE	40
4.1.1	DATA ACQUISITION UNITS (DAU)	42
4.1.2	MICROPROCESSOR CLUSTER (MC)	42
4.1.3	DATA COMMUNICATIONS NETWORK	43
4.1.4	STATION COMPUTER	44
4.2	THE WESPAC SYSTEM	44
4.2.1	THE SYSTEM ARCHITECTURE	45
4.2.1.1	DATA ACQUISTION UNIT (DAU)	45
4.2.1.2	SERIAL DATA LINKS	48
4.2.1.3	PROTECTION CLUSTER (PC)	48
4.2.1.4	DATA HIGHWAY	50
4.2.1.5	STATION COMPUTER (SC)	50
4.2.2	SYSTEM OPERATION	51
4.2.3	SYSTEM HEARTBEAT	52
4.2.4	REDUNDACY	52
4.3	NOOR'S ASCD NETWORK	53
4.3.1	ASSIGNED-SLOT-CSMA/CD PROTOCOL	53
4.4	SUMMARY	56
5.	CONCEPTUAL DESIGN OF THE PROPOSED NETWORK	57
5.1	NETWORK REQUIREMENTS	57
5.1.1	VOLUME OF DATA FLOW	58
5.1.2	RESPONSE TIME	60
5.1.3	SYSTEM AVAILABILITY	61
5.2	THE PROPOSED NETWORK	62
5.2.1	THE ADVANTAGES OF THE NEW APPROACH	65
5.2.1.1	SIMPLICITY	66
5.2.1.2	MODULARITY	66
5.2.1.3	EXPANDABILITY	66
5.2.1.4	MESSAGE PRIORITY	67
5.2.2	PRIORITY CLASSES	67

5.3	THE MESSAGE PRIORITY CARRIER SENSE MULTIPLE ACCESS PROTOCOL	68
5.3.1	THE MPCD PROTOCOL	69
5.4	SUMMARY	73
6.	PERFORMANCE ANALYSIS	75
6.1	CONSTRUCTION OF THE MPCD SIMULATION MODEL	76
6.1.1	SIMULATION CLOCK	77
6.1.2	EVENT QUEUE DISCIPLINE	78
6.2	RANDOM NUMBER GENERATION	79
6.3	SIMULATION MODEL	79
6.3.1	PACKET SIZE	80
6.3.2	DATA PRIORITY LEVEL	81
6.3.3	PERFORMANCE STATISTICS	81
6.3.4	SIMULATION RESULTS	82
6.3.5	COMPARISON TO NOOR'S ASCD NETWORK	85
6.3.6	SUMMARY	86
7.	CONCLUSIONS	89
7.1	MAIN CONTRIBUTIONS	89
7.2	SUGGESTIONS FOR FUTURE RESEARCH	90
REFERENCES		91
APPENDICES		98
A.	LISTING OF THE MPCD MODEL	99
B.	LISTING OF THE SMPL SIMULATION ENVIRONMENT	105

LIST OF FIGURES

	FIGURE	Page
2.1	ARRANGEMENT OF A POWER SYSTEM	7
2.2	THE BASIC ARCHITECTURE OF A DIGITAL RELAY	11
3.1	MAJOR COMPONENTS IN A LAN	21
3.2	KEY CHARACTERISTICS OF LANS'S TRANSMISSION MEDIA	22
3.3	LOCAL AREA NETWORK TOPOLOGIES	25
3.4	LOCAL AREA NETWORK ACCESS CONTROL TECHNIQUES	26
3.5	COMPARISION OF THE CHANNEL UTILIZATION (S) VS LOAD (G) FOR VARIOUS CSMA PROTOCOLS	30
3.6	ETHERNET OPERATION	31
3.7	THE CHANNEL EFFICIENCY VS NO. OF READY STATION OF A ETHERNET	32
3.8	THE BIT MAP PROTOCOL (N=8)	33
3.9	(A) LISTEN MODE (B) TRANSMIT MODE	35
3.10	THE IEEE 802 STANDARD	37
4.1	DATA FLOW IN SUBSTATION	40
4.2	THE HIERARCHICAL ARCHITECTURE OF THE SPC SYSTEM	41
4.3	THE WESPAC SYSTEM ARCHITECTURE	46
4.4	THE DATA ACQUISTION UNIT	47
4.5	THE PROTECTION CLUSTER	49
4.6	THE STATION COMPUTER	49
4.7	COLLISION DETECTION MACHANISM FOR THE NODES ASSIGNED TO THE SAME SLOT	55
5.1	THE CONFIGURATION OF A DISTRIBUTED LAN FOR SUBSTATION PROTECTION AND CONTROL	62
5.2	THE FLOW CHART OF THE MPCD PROTOCOL FOR DATA (i)	72
6.1	CSMA/CD FRAME FORMAT	81
6.2	AVERAGE DELAY FOR VARIOUS LOAD	83
6.3	CHANNEL UTILIZATION AT DIFFERNET LOAD FACTOR	84
6.4	AVERAGE DELAY COMPARISON OF MPCD PROTOCOL AND ASCD PROTOCOL	85
6.5	CHANNEL UTILIZATION COMPARISON OF MPCD PROTOCOL AND ASCD PROTOCOL	87

LIST OF ABBREVIATIONS

τ	END-TO-END PROPAGATION DELAY
B	PACKET SIZE
A/D	ANALOG TO DIGITAL CONVERTER
AI	ANALOG INPUT CARD
ASCD	ASSIGNED SLOT CARRIER SENSE MULTIPLE ACCESS WITH COLLISION DETECTED
ADC	ANALOG TO DIGITAL CONVERTER
CI	CONTACT INPUT CARD
CO	CONTACT OUTPUT CARD
CSMA	CARRIER SENSE MULTIPLE ACCESS
CSMA/CD	CARRIER SENSE MULTIPLE ACCESS WITH COLLISION DETECTED
DAU	DATA ACQUISITION UNIT
DIOB	DISTRIBUTED INPUT/OUTPUT BUS
EOC	END OF CARRIER PACKET
EPRI	ELECTRIC POWER RESEARCH INSTITUTE
I	INTERFACE
I/O	INPUT/OUTPUT
LANs	LOCAL AREA NETWORKS
Mbps	MEGA BITS PER SECOND
MC	MICROPROCESSOR CLUSTER
MFCD	MESSAGE PRIORITY CARRIER SENSE MULTIPLE ACCESS WITH COLLISION DETECTED
ms	MILLI-SECOND
PC	PROTECTION COMPUTER
SC	STATION COMPUTER
VLSI	VERY LARGE SCALE INTEGRATION
WESPAC	WESTINGHOUSE SUBSTATION PROTECTION AND CONTROL SYSTEM

CHAPTER 1
INTRODUCTION

During the past decade, Local Area Networks (LANs) have developed rapidly into a dominant force in the field of computer communications. The major reason behind this rapid development is the vast improvements in the Very Large Scale Integration (VLSI) technology [1]. As the number of computers increases, network designers become more and more interested in connecting them to form a distributed computer network for applications that were once thought impossible. By having different machines perform different tasks, the approach of distributed computing has a variety of advantages [2], including

- sharing expensive resources
- improving productivity
- adding functionality
- exchanging data between computers

However, not until recently, LANs have been used mainly in office and business environments for such application as office automation. During the past few years, numerous studies have been conducted using LANs in a real-time control environment. One of these studies is the digital protection and control scheme at power sub-stations [3].

1.1 REVIEW OF CURRENT RESEARCH

As early as 1978, a study of distributed computer networks for application to the control and protection of electric power substation was published by Rummer and Kezunovic [4,5], where the needs and the required functions of the digital protection and control scheme have been identified and defined. A Distributed processing Microprocessor-based Hierarchically-structured (DMH) system was also presented in the paper.

About the same time, a dedicated digital system, developed jointly by Kansi Electric Power Company and Mitsubishi Electric Corporation in Japan [6], used thirteen microprocessors to perform all the control and protective functions at a substation. The fundamental concept underlying this system is a three data channel structure to manage all the signal flows for control and protection. Satisfactory results have been reported from field tests, and promising results from these two early studies led to subsequent studies.

In the early 1980's, the Electric Power Research Institute (EPRI) initiated and sponsored numerous research and development projects in integrated microprocessor based power substation control and protection systems [3,7-9]. The aims of these studies were to reduce the life time costs of substation control and protection functions while achieving performance and featuring benefits of new digital approaches by employing recent advanced electronics technology. The Westinghouse substation protection and control (WESPAC) system is the direct result of these studies [10]. The WESPAC system, an integrated modular system built on the state-of-the-art digital communications technology, provides protection, control, and monitoring at the substation. Test systems have been operated at the Branchburg substation of the Public Service Electric and Gas Co. (PSE&G) of New Jersey since September 1984 and at the Deans 500/230 KV substation of PSE&G since early 1986 [11].

A study of a microcomputer network for real-time processing requirements in a substation was done by Noor in October 1982 [12]. In his work, Noor proposed to use an assigned-slot-CSMA/CD (ASCD) network protocol for the real-time substation protection and control system. An analytic model and a simulation model were developed to predict the performance of the ASCD network. Both the Westinghouse WESPAC and Noor's ASCD network are discussed in detail in chapter 4.

Several other studies [14,15] have addressed the problems of a LAN in real-life control and protection environment. One of the more interesting studies was done at the University of Regina, together with the Saskatchewan Power Corporation, they conducted a study on a Local Area Network for an electric power substation in September 1984 [13]. This study, supported in part by Natural Science and Engineering Research Council of Canada, investigated various LAN architectures and protocols for the suitability of an automated protection and control substation. A token passing ring was proposed for this application. Simulation results show that the performance of the network meets the data communication requirements within a substation. These studies were general in scope.

1.2 RESEARCH APPROACH

The objective of this research is to propose a local area network (LAN) protocol with performance best suited for a real-time environment in a digital protection and control power substation. The approach taken towards the development of this research is to develop a simulation model for the proposed protocol. Numerical results from the simulation are then analyzed to predict the network performance.

Prior to the selection of the proposed network, a detailed study of the digital protection and control scheme is carried out to determine the substation's requirements. A LAN protocol is then proposed for substation application on the basis of the defined requirements.

A comparative study of the network performance is then performed between the proposed protocol and the previously mentioned Noor's ASCD protocol [12] to determine which protocol is better suited for application to substation protection and control.

The major concerns in the performance study are:

1) Channel utilization (throughput)

2) response time (speed requirement)

3) data rate

4) size of the network

5) type of data

1.3 CONTENTS OF THE THESIS

This study begins with a brief history of the development of digital protection and control of the power apparatus at the substation. This chapter also introduces the fundamental concepts of digital protection and control. A study is conducted to identify and define the substation functions and requirements. Four major functions of the network are discussed: protection, control, monitoring and recording/displaying.

The concept of LANs is described in chapter 3, which includes the motivation, the topology and the network protocol. Two major LAN protocols are reviewed, the Carrier Sense Multiple Access (CSMA) protocol and the Token ring protocol.

Chapter 4 deals with the architectural structure of the substation protection and control system. A three level hierarchical structure is used to illustrate the digital protection and control system. The functions of each element in the hierarchy are reviewed. In addition, the architectural structure of the WESPAC system as well as the Noor's ASCD protocol are also discussed in detail.

In chapter 5, the proposed protocol, message priority CSMA/CD (MPCD), is presented. A detailed description of the MPCD protocol is outlined. The MPCD protocol is based on the principle that access right to the channel (network) is exclusively granted to ready messages of the current highest priority level. An analysis is performed to classify the substation data into three priority classes, fault data, protection data and system data. The advantages of the MPCD protocol are also reviewed in this chapter.

Chapter 6 is devoted to the simulation modeling of the MPCD protocol. A performance analysis is performed to examine the numerical results obtained from the simulation. The throughput-delay characteristic of the MPCD protocol is derived. These results are also compared with the Noor's ASCD results.

Finally, on the basis of this study, conclusions and suggestions for further research are presented in chapter 7.

CHAPTER 2

DIGITAL PROTECTION AND CONTROL IN POWER SUBSTATIONS

The idea to use digital computers for real-time applications such as protection and control of power system equipment was first started in the late 1960's. At the time, several studies were done to examine the feasibility of using a mini-computer as a dedicated digital protective relay to detect and locate faults in power systems. Since then, there has been considerable interest and research conducted in this area.

The development of low cost, but powerful microcomputers provided an important element needed for the cost-effective implementation of computer protection and control of power systems. A number of protection algorithms have been proposed and developed [17-22]. The latest trend is to interconnect these newly designed protective relays and control devices to form a data communications network (local area network) within the power substation [3-14].

In this chapter, the development of digital protection and control scheme is described. The advantages of the digital relays are also discussed.

2.1 POWER SYSTEM FAULTS & PROTECTIVE RELAYS

Modern electric power systems are dependable, and ready to deliver energy to the customer without any interruption. Protective relays play an important role in assuring this continuous service. Occasionally, power systems do experience faults and abnormal

operating conditions. These are identified by the protective relays. Once a fault is detected, the relays initiate corrective action to minimize any service interruption.

A power system can be considered as a chain consisting of generators, power transformers, switchyard, transmission lines, distribution circuits, and utilization apparatus (see figure 2.1).

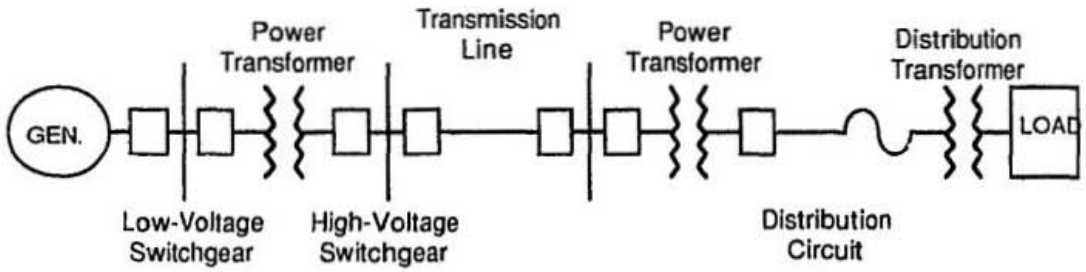

Figure 2.1 Arrangement of a power system

There are a number of causes for the failure or breakdown of these components. Faults or short circuits can occur between individual phase wires or coils and between a phase wire or coil to ground as a result of breakdown of the insulation protecting them. The resulting electric arc which contains considerable power can cause severe damage in a very short time. These faults or short circuits are caused mainly by insulation failure, but may also induced by such things as voltage surges, overloading with subsequent overheating of the equipment, and abrasion due to expansion and contraction. Transmission line faults can be caused by some factors such as wind, ice and sleet, large birds bridging the insulators, lightning, swinging tree limbs, and crane booms, etc. Other abnormal conditions which impair a component's function in the power system are overheating of bearings, over or under speed, and reverse phase sequence, etc.

2.1.1 THE FUNCTION OF PROTECTIVE RELAYS

Relays are placed in the power system to ensure uninterrupted service to the customers. They do so by avoiding equipment damage or by limiting it to the single unit that may be in trouble. The relays locate the fault and trip the circuit breakers which will break off any links to the defective apparatus, thereby isolating it.

The protective relays obtain the necessary information to locate a fault in the form of currents and voltages from instrument transformer (transducers or interfaces) which are located on specific parts of the power system being protected. This piece of information is then relayed to the circuit breakers in the form of a tripping pulse (signal). Finally, the circuit breakers isolate the defective apparatus by interrupting the flow of current from all sources.

2.1.2 TYPES OF PROTECTIVE RELAYS

Protective relays basically consist of an operating unit and a set of contacts. The operating unit takes on the information from the instrument transformers in the form of currents and voltages, performs a comparison operation to determine if there is a fault, and converts the result into a motion of contacts accordingly. When they close, the contacts either actuate a warning signal or complete the trip circuit of a circuit breaker, which in turn completes the isolation of the faulty element by interrupting the flow of current into that element.

Conventionally, there are two major types of relays available, electromechanical relays and solid state relays. The electromechanical relays are the older of the two types, but are still widely used in the industry. The solid state relays were first introduced in the 1950's. However, they were not widely accepted until the late 1960's.

Solid state relays react considerably faster than electromechanical relays. The solid state circuits are designed to provide various functions such as level detection, phase angle measurement, amplification, pulsing, squaring, timing and others. These analog and partly digital circuits react instantaneously to the inputs of current and voltage so as to supply the proper outputs for the required characteristics.

2.1.3 THE SHORTCOMINGS OF CONVENTIONAL RELAYS

Despite generally performing adequately, there are few deficiencies and problems associated with the conventional electromechanical and solid state relays. The following lists the salient shortcomings of these relays:

Silent sentinel type :

Since the conventional relay hardware is in protracted idle state until a fault is detected, they may fail without warning. Most utilities have conducted periodic maintenance programs in which the relays are tested. These programs are costly, and may still fail to catch many problems until after the failure. Futhermore, such periodic human tampering may cause additional damages and problems.

High cost :

The cost of conventional relays continued to increase during the last twenty years, while the cost of digital devices and microcomputers has been decreasing during the same period. Thus economics make the development of digital relays even more attractive.

Dedicated type :

All conventional relays are of the dedicated type. Different protection functions require different types of relays.

2.2 THE DIGITAL RELAYS

There has been a considerable amount of interest in digital protection of power apparatus since the late 1960's. In 1969, dedicated digital relays using minicomputers were proposed by G. D. Rockefeller [24], who defined and examined protection by using a modular approach for all types of station equipment. His proposal is often referred to as the foundation of the concept of digital relaying.

Digital relays are desirable since they permit continuous monitoring and self-checking. They also have the ability to consolidate the logical functions of many devices into one processor unit, therefore avoiding duplication in situations where many separate pieces of equipment use identical inputs or perform similar functions. These lead to the possibility of the integrated automated substation.

The rapid development of microcomputers has brought about novel and low-cost possibilities for the development of digital relays. The high capability of presently available microcomputers are such that all the digital relaying functions performed by the minicomputers of the late 1960's can now be done by these microcomputers. As a result, many research works have been conducted on the specific hardware and software techniques for microcomputer based relays [3-14].

2.2.1 THE CONCEPTS OF DIGITAL PROTECTION SCHEME

The basic architecture of a digital relay is outlined in figure 2.2. The blocks include a data acquisition unit (DAU), an analog to digital converter (ADC), a microcomputer and an input/output subsystem.

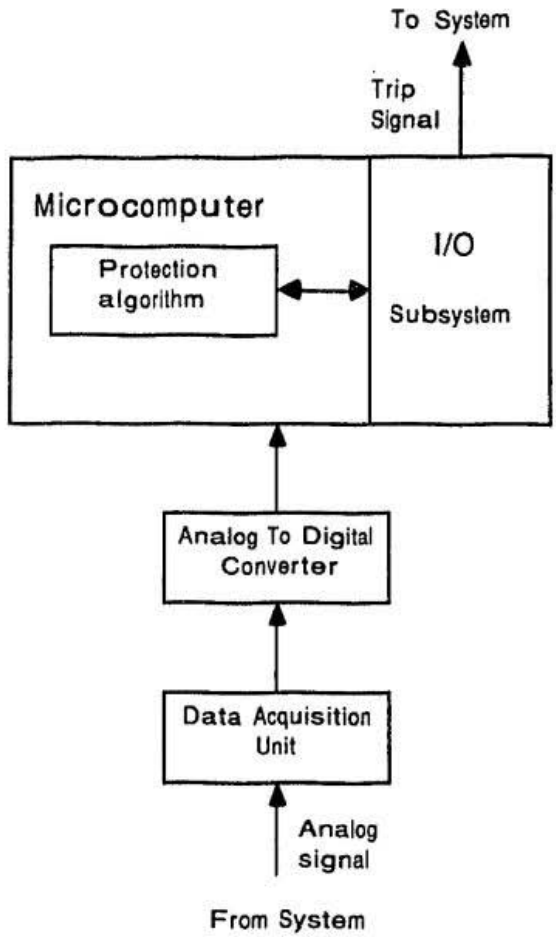

Figure 2.2 The basic architecture of a digital relay

The DAU receives analog signals that represent power system currents and voltages; these signals are converted into digital form (bits) by the ADC. The digital information is then passed on to the microcomputer where an appropriate relay program (protection algorithm) is executed based on the input digital data to determine the system condition. The decisions of the relay are conveyed to the system through the input/output subsystem.

The heart of the digital relay system is the relay program. Various algorithms have been proposed and implemented for all types of protection over the years [17]. Promising results have been obtained from these studies. The following lists some of the relay algorithms based on the protection apparatus:

Transmission Line [25,26] :

- Fourier Algorithm
- Correlation Technique
- Kalman Filtering Approach

Transformer [17,22,26] :

- Fourier Algorithm
- Rectangular Transform Approach
- Walsh Algorithm
- Haar Function Approach
- Finite Impulse Response Approach
- Least Square Curve Fitting Algorithm

2.2.2 THE ADVANTAGES OF DIGITAL RELAYS

The main reasons for the development of digital relays are their economy, reliability, flexibility, and improved performance over the conventional relays. Some of the salient advantages of digital relays are :

1) Faster breaker tripping time : This gives security against undesired operations comparable to conventional relays.

2) More reliable : Since the hardware of the digital relay is in frequent use, this increases the confidence level in the serviceability of the relay. Also, the digital relay can be designed to monitor itself and alert operators in the event of equipment failure.

3) More economical : Costs of microcomputers are decreasing rapidly while their power is increasing.

4) More flexible : Most of the hardware is the same for all types of digital relays, only their softwares are different. This creates a great deal of flexibility for the relay.

5) Automation : There is the possibility of integrating the digital relays into a computer network to form an automated (unmanned) protection and control substation.

2.3 INTEGRATED MICROCOMPUTER BASED SUBSTATION PROTECTION AND CONTROL SYSTEM

Computer applications in protection and control of power systems have been affected by the developments in digital relays and microcomputer systems. A number of projects related to substation control and digital protection have been reported by utility companies and research organizations throughout the world since the late1970's. The major objective of these studies is to develop a digital protection and control substation concept using the latest technology. The new approach enables process information from the digital relays of the power system converted to digital form and multiplexed onto a single trunk or highway leading to the control area, where station computers record data, control process operations, and provide data records for operators. This provides the potential for more sophisticated capabilities at lower installation cost.

The concepts behind the development of the microcomputer network to meet the objective of the processing requirements of a transmission level substation are described in this section. A number of functions and requirements of the network are specified. These functions have formed the basis for the later analysis of the network.

2.3.1 THE NEEDS FOR TRANSMISSION PROTECTION AND CONTROL

Before describing the functional requirements for the substation protection and control system, it is necessary to define the need for the system.

2.3.1.1 NEED FOR FAST RESPONSE AND SYSTEM DEPENDABILITY

The most important factor in the protection and control scheme is the dependability of the system for a wide range of power system effects. Rapid response to events by subtle differences in measured signals is required for many of the functions. Sophisticated relaying and control are required to maintain overall reliability of the power system.

2.3.1.2 NEED FOR CENTRALIZED CONTROL FUNCTIONS

To achieve the discrimination which is essential for effective protection and control, a variety of sophisticated sensing functions must respond in a co-ordinated way. This implies that all information from the switchyard must be collected at a central location, the control center.

2.3.1.3 LARGE DISTANCES

The physical size of a typical Extra High Voltage (EHV) station is perhaps its most striking feature. The concern is that the utility must run many wires hundreds or even thousands of feet between switchyard apparatus and the control center for signal transmission and for power equipment control. What is needed to solve the problem is an integration of functions so that the equipment requires a minimum of interconnections, and therefore, shorter wires.

2.3.1.4 WITHSTANDING THE ELECTROMAGNETIC ENVIRONMENT

The power apparatus produces an environment of severe transient Electromagnetic Interference (EMI) which threatens any low level signal electronic system (such as digital devices) installed in the substation.

2.3.1.5 NEED FOR EVENT RECORDING

The operation of control or protection elements has a direct effect on the security of the transmission network. Therefore, recording equipment must monitor the behavior and operation of these critical devices. This recording gear must be automatic to capture the response of important events such as faults, power swings, and switching. This information is important for later analysis which leads to further adaptations of operating practice and control system.

2.3.1.6 NEED FOR UNATTENDED OPERATION

The remoteness of some substations, the cost of having an operator on site, and the need for the speed of automatic control actions have led to the desire for unmanned substations. For each substation, all control activities must be performed either automatically, or at the command of a remote system operator via the Supervisory Control And Data Acquisition (SCADA) system [27]. It would be also desirable to have some sort of self-diagnosis or internal and external monitoring so that system maintenance personnel are alerted when something does go wrong.

2.3.1.7 NEED FOR SYSTEM-WIDE CONTROL

Comprehensive centralized monitoring and control of the system is essential to modern power system. Data from around the station are collected for detailed analysis and modeling. The results lead to reconfiguration or readjustment commands which improve the security of the system. System-wide control functions must be supported by interface equipment which supplies digested data to the control center at the substation.

2.3.1.8 NEED FOR PERIODIC MODIFICATION

It is uncommon to design and build a major transmission substation from scratch to its ultimate configuration. On the contrary, it typically begins as a node for only a few lines, and evolves in steps over a period of years into a major junction with many lines and transformers linking different levels of the grid. Each expansion requires additional control and protection equipment. Therefore, it is desirable to have a system that is flexible enough to facilitate such changes with a minimum of effort and expense.

2.3.2 SUBSTATION FUNCTIONAL REQUIREMENTS

A function list was compiled in the ERPI's projects [7-9] to define the functions that must be supported by the substation protection and control system. These functions fall under four general categories:

2.3.2.1 PROTECTION FUNCTIONS

These functions initiate high speed tripping of circuit breakers at each terminal when a fault is detected anywhere along the protected zone of the corresponding component. They are:

- line fault protection
- transformer fault protection
- bus fault protection
- shunt reactor protection
- breaker failure protection
- transfer tripping

Line protection consists of pilot and non-pilot protection with delayed remote backup protection. Transformer trip logic includes over-excitation, differential overcurrent, and time overcurrent protection. Transformer restrain logic includes magnetising inrush current protection.

2.3.2.2 AUTOMATIC CONTROL FUNCTIONS

Automatic control functions are :

- local control of voltage and VAR flow
- load shedding
- automatic reclosing
- synchronism checking and synchronizer closing
- automatic switching sequences

2.3.2.3 MONITORING FUNCTIONS

The monitoring functions include supervision and periodic testing of the integrity of system components. The monitoring functions are :

- pilot and transfer trip channel monitoring
- load monitoring and out of step protection
- monitoring and control of breakers and switches
- diffferential measurement check
- transformer overload and tap position monitoring
- self checking

2.3.2.4 RECORDING AND DISPLAYING FUNCTIONS

This category groups all the functions which pertain to the interfaces between the protection and control system and the external world - SCADA masters, local and remote operators as well as the information displayed to them. These functions are:

- local man-machine subsystem
- remote SCADA interface
- alarming
- data logging
- revenue metering
- recording and indication of sequence of events
- oscillography
- line fault location estimation

2.3.3 SYSTEM RELIABILITY

A well protected substation needs an extremely high degree of reliability for the entire chain of protection, including the network link. Unlike many computer network systems, the system downtime must be low, the substation protection and control system must ensure practically instantaneous recovery from faults due to element failure. Therefore, redundancy of each protective functions is required by the system to obtain a high degree of reliability.

2.3.4 SUMMARY

The concept of protective relays has been outlined including the shortcomings of the conventional relays.

The development of digital relays using microcomputers was presented and their advantages were described. In addition, the concept of digital protection scheme has also been described.

The design goals of the substation protection and control system have been established. A number of functions have been defined and their requirements are specified. These functions form the basis for later design consideration. The detail design of the required network is described in chapter 5.

CHAPTER 3

LOCAL COMPUTER NETWORKING

The idea of local computer networking was first introduced in the 1960's as an attempt to find new technologies for telephony [28,29]. The initial developments failed due to high cost and unreliability of contemporary electronics. During the1970's, the thought of local computer networking was picked up again to link minicomputers together by a number of research laboratories such as Xerox's Palo Alto Research Center (PARC) and the University of Hawaii.

By the mid 1970's, several network architectures have been proposed which include the U.S. Department of Defense's ARPA network [30,31], the University of Hawaii's Aloha network [32,33], Mitre's MITRIX [34], Bell Telephone Laboratory's Spider [34], the University of California, Irvine's Distributed Computing System (DCS) [35] and Xerox's Ethernet [36]. The trends of this rapid development of local computing networking continued and developed into today's Local Area Network (LAN) technology.

This chapter introduces the LAN technology. It characterizes the LANs by their topologies and transmission media. Various medium access control protocols including CSMA/CD and token ring are described.

3.1 LOCAL AREA NETWORKS (LANS)

Local area network is distinguished by the area it encompasses; it is geographically limited within a distance of a few tens of kilometers. They differ from large

computer networks in several ways. The main difference is that large networks are often economically and legally restricted to use the public telephone network, regardless of its technical suitability. On the contrary, most LANs are privately owned and operated. They can use high bandwidth cables which leads to numerous advantages such as high data rate (0.1 - 100 Mbits/s, Mbps) and low error rate (10^{-8} to 10^{-11}).

A LAN usually contains four major components (see figure 3.1), which serve to transport data between end users.

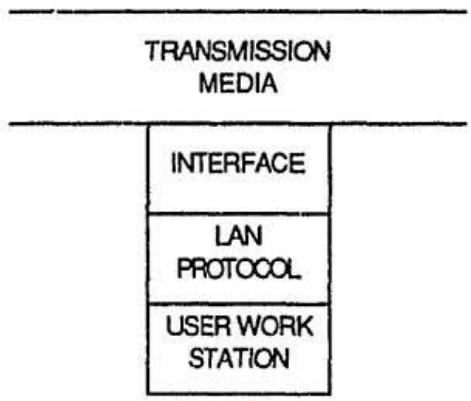

Figure 3.1. Major components in a LAN

3.1.1 TRANSMISSION MEDIA

LANs's transmission media can take several different physical forms, twisted pair wire, coaxial cable and fiber optic. Figure 3.2 summarize key characteristics of these media. Two major factors are usually considered when choosing a transmission medium: the speed requirement and the distance of the transmission.

Transmission Medium	Data Rate	Bandwidth	Repeater Spacing
Twisted Pair	4 Mbps	250 KHz	2-10 km
Coaxial Cable	500 Mbps	350 KHz	1-10 km
Fiber Optic	2 Gbps	2 GHz	10-100 km

(a) Point-to-point configuration

Transmission Media	Signalling Techniques	Maximum Data Rate (Mbps)	Max. Range @ Max. Data Rate (km)	Practical No. of Devices
Twisted Pair	Digital	1-2	Few	10's
Coaxial Cable (50 Ω)	Digital	10	Few	100's
Coaxial Cable (75 Ω)	Digital	50	1	10's
	Analog with FDM	20	10's	1000's
	Single Channel Analog	50	1	10's
Fiber Optic	Analog	10	1	10's

(b) Multipoint configuration

Figure 3.2. Key characteristics of LANs's transmission media

Twisted pair wiring is one of the most common communications transmission media. Due to its low bandwidth, twisted pair wiring is used mainly for low speed transmission. Data rates of up to a few Mbps can be achieved. One major disadvantage of the twisted pair wire is its susceptibility to interference and noise, including cross-talk from adjacent wires. These effects can be minimized with proper shielding. Because of its relatively low cost, the twisted pair wire is the most cost effective choice for low traffic requirements.

Coaxial cable is the most commonly used LAN's transmission medium. It possesses high capacity, a good noise immunity and a low error rate. It provides a higher performance,

can support a larger number of devices, and can span greater distances than the twisted pair wire.

Two transmission techniques, baseband and broadband, can be employed on a coaxial cable. The main difference is that baseband uses digital signalling and supports a single data channels, whereas broadband uses analog signalling in RF range and supports multiple simultaneous data channel. In addition, baseband transmission is bi-directional while broadband is uni-directional. Additional information on LAN baseband and broadband transmissions can be found in reference [37].

The transmission medium with the greatest bandwidth and noise immunity is fiber optic. It has a higher potential capacity than coaxial cable, and a number of advantages over both coaxial cables and twisted pair wire, including light weight, small diameter, low noise susceptibility, and practically no emissions. However, it has not been widely used in the past due to high cost, difficulty of installation and other technical limitations [38]. From a technical point of view, point-to-point topologies like the ring are feasible. Multipoint topologies like the bus/tree with taps at each node are not practical. With recent rapid development in fiber optic technology, the high cost for the fiber optic interface has fallen into a level that makes this medium competitive [51]. It is likely that fiber optics become the preferred transmission medium for LANs in the future.

3.1.2 INTERFACE

The interface between the transmission path and protocol logic can take several forms. It may be a single cable television (CATV) tap, infra-red diodes for infra-red path, microwave antennas, or complex laser-emiting semiconductors for fiber optics. Some LANs

provide regenerative repeaters at the interface, other use the interface as buffers for data flow and/or simple connections.

3.1.3 PROTOCOL CONTROL

The protocol control logic component controls the LAN and provides for the end user's access onto the network. Details of the LAN protocols are discussed in section 3.3.

3.1.4 END USERS

End users include any data communication devices that could communicate over the transmission medium, they can be :

- computers
- terminals
- peripheral devices
- sensors (temperature, humidity, security alarm sensors, etc.)
- telephones

3.2 TOPOLOGIES

LANs are often characterized in terms of their topologies, three of the most common topologies are shown in figure 3.3.

The bus topology is characterized by the use of a multiple access, broadcasting medium. Because all devices share a common communications medium, only one device can transmit at any instant. A station wishing to transmit waits for its turn and then sends data out onto the bus network in the form of a packet, which contains both the source and destination addresses as well as the data. Each station monitors the transmission medium and copies the packet addressed to it.

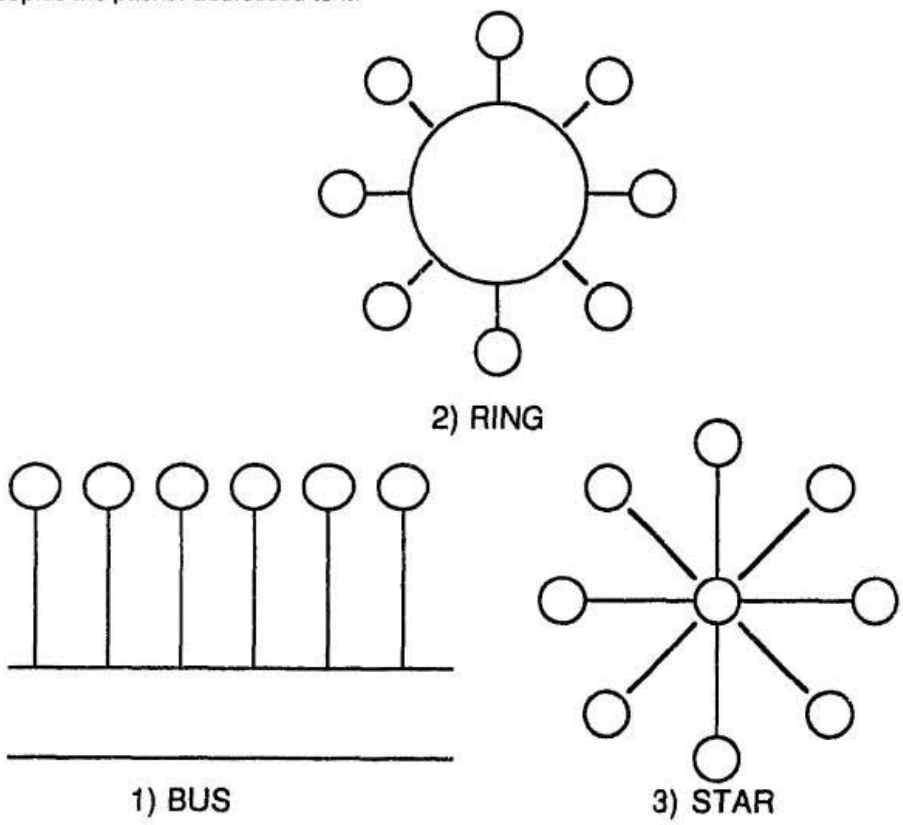

Figure 3.3. Local area network topologies

The ring topology consists of a closed loop, with each node attached to a repeating element. Data circulate around the ring on a series of point-to-point data links between repeaters. Like the bus topology, it transmits a packet containing source and destination addresses and data. As the packet circulates, the destination node copies the data into a

local buffer. The packet continues to circulate until it returns to the source node, providing a form of acknowledgement.

In the star topology, a central switching element is used to link all the devices in the network. A station wishing to transmit data sends a request to the central switch for a connection to a destination station, and the central element uses circuit switching to establish a dedicated path between the two stations. Once the circuit is setup, data are exchanged between the two stations as if they were connected by a dedicated point-to-point link.

3.3 LANS PROTOCOL

The most common access protocols for LANs are categorized in figure 3.4 in according to their signalling techniques. Two of the most popular access protocols, carrier sense multiple access (CSMA) and token ring are discussed here.

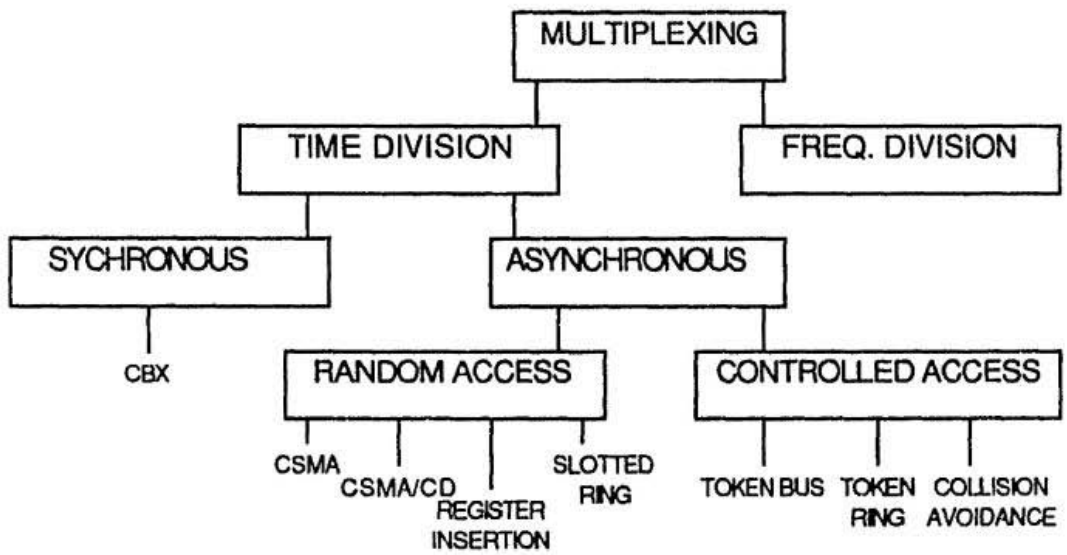

Figure 3.4. Local area network access control techniques

3.3.1 CARRIER SENSE NETWORK

In carrier sense networks, a station listens to the transmission before sending a packet. It can then base its action on whether the channel is busy or not. Protocols in which stations listen for a carrier (i.e. a transmission) and act accordingly are called carrier sense protocols.

All of the carrier sense protocols discussed below are based on the following assumptions [39] :

1) All packets are of constant length

2) There are no errors, except those caused by collisions

3) There is no capture effect

4) The random delay after a collision is uniformly distributed and is large compared to the packet transmission time

5) Packet generation attempts (new ones plus retransmissions) form a Poisson process with mean G packets per packet time

6) A station may not transmit and receive simultaneously

7) Each station can sense the transmissions of all other stations

8) The propagation delay is small compared to the packet transmission time, and identical for all stations

9) Sensing the state of the channel can be done instantaneously

3.3.2 CSMA PROTOCOL

3.3.2.1 1-PERSISTENT CSMA

The 1-persistent CSMA protocol is the simplest carrier sense protocol. In this protocol, a station with ready data senses the channel and proceeds as follows.

1) If the channel is sensed busy, the station waits and continually senses the channel until it becomes idle.
2) When the station senses an idle channel, it transmits the data. If a collision is detected, the station waits a random amount of time and repeats the algorithm.

The protocol is called 1-persistent, because the station transmits with a probability of 1 whenever it finds the channel idle.

The propagation delay (τ) has an important effect on the performance of this protocol. There is a small chance that just after a station (A) begins sending, another station (B) will become ready to send and sense the channel. If A's signal has not yet reached B, the latter will sense an idle channel and will also begin sending, resulting in a collision.

Even if the propagation delay is zero, collision will still occur if A and B become ready in the middle of a third station's (C) transimission, both will wait until the transmission ends and then both will begin transmitting simultaneously, collision thus takes place.

3.3.2.2 NON-PERSISTENT CSMA

The non-persistent CSMA protocol is an improved 1-persistent CSMA protocol. In this protocol, a station senses the channel before sending. If the channel is sensed idle, the station sends the data. However, if a transmission is already in progress, the station does not continually sense the channel for the purpose of seizing it immediately upon detecting the end of previous transmission. Instead, it waits a random period of time and then repeats the algorithm. This algorithm gives a better channel utilization and longer delays than the 1-persistent CSMA.

3.3.2.3 P-PERSISTENT CSMA

Unlike the previous two protocols, P-persistent CSMA applies only to the slotted channel. In a slotted channel, time is divided into discrete intervals, each interval corresponding to one packet. A ready station senses the channel and operates as follows.

1) If the channel is idle, it transmits with a probability of P. With a probability of Q = 1 - P, it defers until the next slot. If that slot is idle, it either transmits or defers again, with the probabilities P and Q, respectively. This process is repeated until either the packet has been transmitted or another station has begun transmitting. In the latter case, it acts as if there has been a collision (ie. it waits a random amount of time and starts again).

2) If the station initially senses the channel busy, it waits until the next slot and applies the above algorithm.

The performances of these three CSMA protocols, as well as some other well known protocols, are plotted as the channel utilization (S) versus load (G) curve in figure 3.5 [39]. The load G is defined as the mean arrival packet (a Poisson process) per packet time.

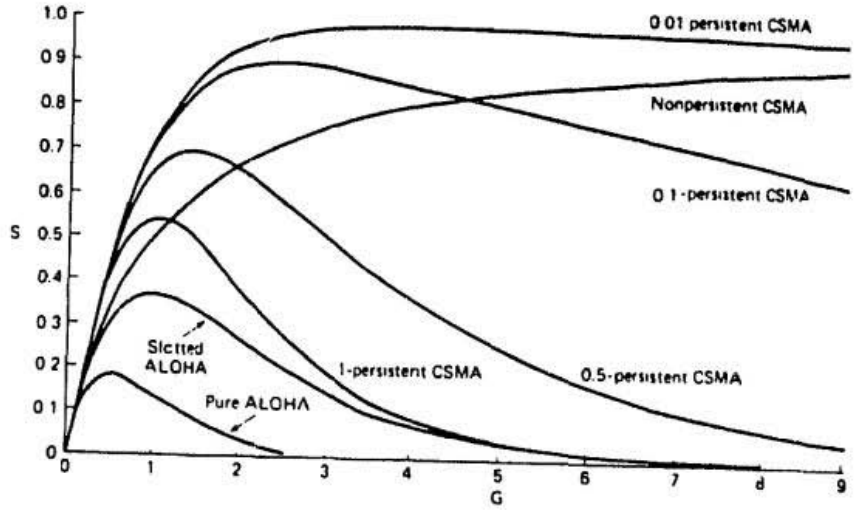

Figure 3.5. Comparision of the channel utilization (S) VS load (G) for various protocols [39]

3.3.2.4 CSMA WITH COLLISION DETECTION (CSMA/CD)

CSMA/CD is the most commonly used medium access control protocol for bus topologies. In this protocol, all stations continue to listen to the cable while transmitting. If a collision is detected, they immediately terminate transmitting the packet and transmit a brief jamming signal to ensure that all stations know there has been a collision. After transmitting the jamming signal, each station waits a random amount of time, and repeats the algorithm. This strategy greatly reduces the amount of time wasted on data collision.

Ethernet, the original baseband version of CSMA/CD, was proposed by the XEROX in 1976 [36]. Each ethernet's station monitors the cable (ether) during its own transmission, terminates transmission immediately if a collision is detected. The operation of the

ethernet is illustrated in figure 3.6. At time t_0, a station has just finished transmitting its packet. Any other station having a ready packet may now attempt to send. If two or more stations decide to transmit simultaneously, there will be a collision. Each will detect the collision, abort its transmission, wait a random amount of time, and try again, assuming that no other station has started transmitting in the meantime.

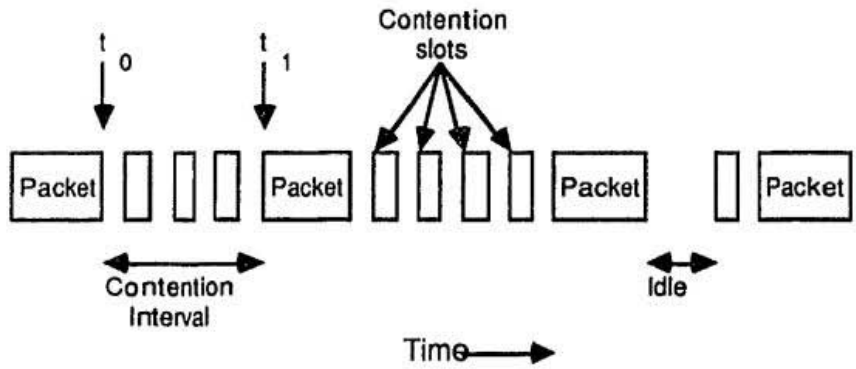

Figure 3.6. Ethernet operation

If τ is the end-to-end propagation delay, it will take a maximum time of 2τ to detect a collision. The contention interval is therefore equivalent to 2τ. This is the worst case time for a station to realize that a collision is detected after a packet is sent.

It is important to realize that collision detection is an analog process. The station's interface hardware must listen to the cable while it is transmitting. If what it reads back is different from what it is putting out, it knows a collision is occurring.

In figure 3.7, the channel efficiency of an ethernet is plotted against the number of ready station for $\tau = 5$ μs and a data rate of 10 Mbps [39].

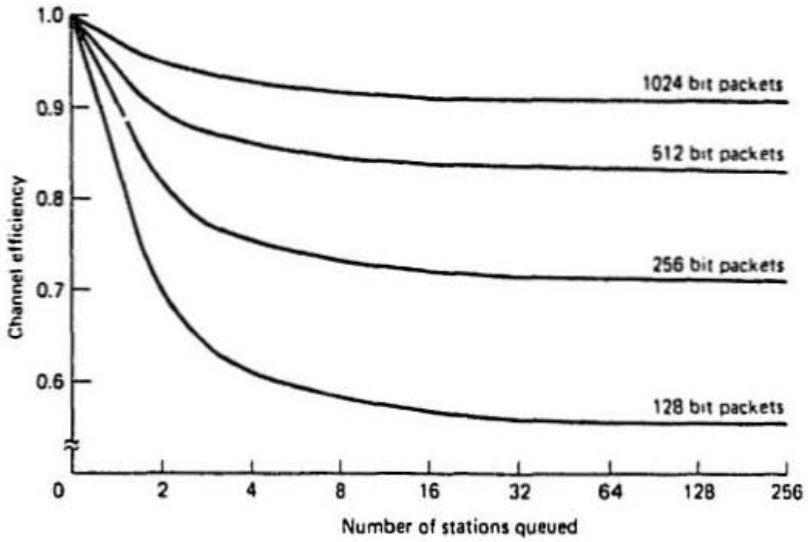

Figure 3.7. The channel efficiency vs no. of ready station of a ethernet [39]

3.3.2.5 COLLISION-FREE PROTOCOLS

Although collisions do not occur on the ethernet once a station has seized the channel, they can still occur during the contention period. These collisions do adversely affect the network performance, especially when the cable is long and the packets are small. Some protocols do resolve the contention for the channel without any collisions.

These collision-free protocols all require some kind of reservation scheme to determine which station will get access to the channel next. A reservation period and some priority functions are usuallly invovled with these protocols.

Bit map protocol is one such collision-free protocol. In this protocol, each reservation period consists of exactly N slots. If station 0 has a packet to send, it transmits a logic high during the first slot (0). No other station is allowed to transmit during this slot. Regardless of what station 0 does, station 1 gets the chance to transmit a "1" during slot 1 when it has a packet to send. In general, station j may announce the fact that it has a packet to send by inserting a "1" into the slot j. After all N slots have passed by, each station has a complete knowledge regarding which stations wish to transmit. At that point, they begin transmitting in numerical order (see figure 3.8). Since every station agrees on which station goes next, there will never be any collisions. After the last ready station has transmitted its packet, another N bit reservation period began. Note that if a station becomes ready just after its bit slot has passed, it must remain silent until everyone has had his say and the bit map (reservation slot) comes around again.

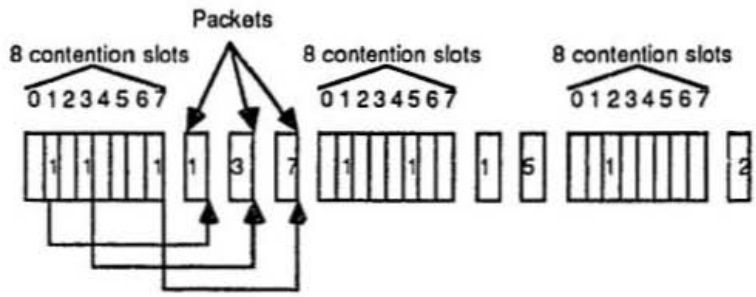

Figure 3.8. The bit map protocol (N=8)

The main drawback of this protocol is that under low load condition, a station must always wait for the current scan to be finished before it may transmit, thereby creating a lot of overhead for the transmission. The tradeoff is between the overhead reservation period and the collision time. One special characteristic of this protocol is the asymmetry with respect to station number; high numbered stations get better services than low numbered ones. This characteristic can allow priority functions to be implemented on the network.

3.3.3 RING NETWORKS

A carrier sense network is basically a passive, electrically connected cable onto which all stations tap, whereas a ring network is actually a series of point-to-point cables between consecutive stations. Also, the interfaces used on ring networks are active rather than passive.

A major issue in the design and analysis of ring networks is the physical length of a bit. If the data rate of the ring is R bits, a bit is emitted every 1/R μs, with a signal propagation speed of X meters/μs, each bit occupies X/R meters on the ring.

A number of different algorithms have been proposed for controlling access to the ring. In the next section, token ring, the most common ring protocol, is discussed.

3.3.3.1 TOKEN RING

The token ring protocol is based on the use of a special bit pattern, called the token, that circulates around the ring [28].

When all stations are idle, the token is labeled as a "free" token (for example, 11111111). A station wishing to transmit waits until it detects the token passing by, then alters the bit pattern of the token from "free" to "busy" token (11111110). Immediately after the the token has been transformed, the station making the transformation is permitted to begin transmitting.

Each arriving bit from the network must be stored in a local buffer, and then a new bit is generated. Therefore, one bit delay is imposed in each ring interface. If there are many stations on the ring, the total effect of these 1 bit delays has an important impact on the performance of the ring. Another implication of the token ring design is that the ring itself must have a sufficient delay to contain a complete token to circulate when all stations are idle.

Ring interfaces operate in two modes: listen and transmit (see figure 3.9). In the listen mode, the inputs are simply copied to the output with a 1 bit delay. In transmit mode, which is entered only after the token has been seized, the interface breaks the connection between input and output, entering its own data onto the ring. To be able to switch from listen mode to transmit mode in 1 bit time, the interface usually needs to buffer one or more packets itself rather than having to fetch them from the station in a short period of time.

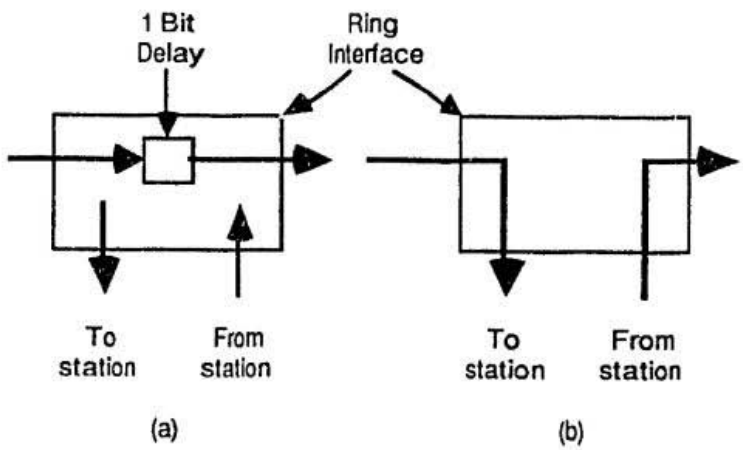

Figure 3.9. (a) Listen Mode (b) Transmit mode

As bits that have propagated around the ring come back, they are removed from the ring by the sender. The sending station can either save them, to compare with the original data to monitor ring reliability, or discard them. Then the sender will regenerate the "free" token to signal that the ring is now idle.

When traffic is light, the token will spend most of the time idly circulating around the ring. Occasionally, a station will convert it to a busy token, followed by a packet and then a new token. However, when the traffic is heavy, so that there is a queue at each station, as soon as a station finishes its transmission and regenerates the free token, the next station down stream will see and remove the token. In this manner, the permission to send rotates smoothly around the ring, in a round-robin fashion.

3.4 LAN STANDARDS

The key to the development of the LANs market is the availability of a low cost interface. This requirement as well as the complexity of the LANs protocols, dictates a Very Large Scale Integration (VLSI) solution. However, Integrated Circuit (IC) manufacturers are reluctant to commit the necessary resources unless there is a high volume market. A LAN standard would ensure that volume and it must also enable equipment of a variety of manufacturers to intercommunicate. This is the rationale behind the IEEE 802 standard which was developed by the IEEE committee on LANs in 1983 [40]. The IEEE 802 standard is in the form of a three-layer communications architecture (see figure 3.10). These three layers, known as logical link control, medium access control and physical, are equivalent to the functionality of the lowest two layers (data link and physical) of the International Organization for standardization (ISO) reference model which developed by the CCITT [39].

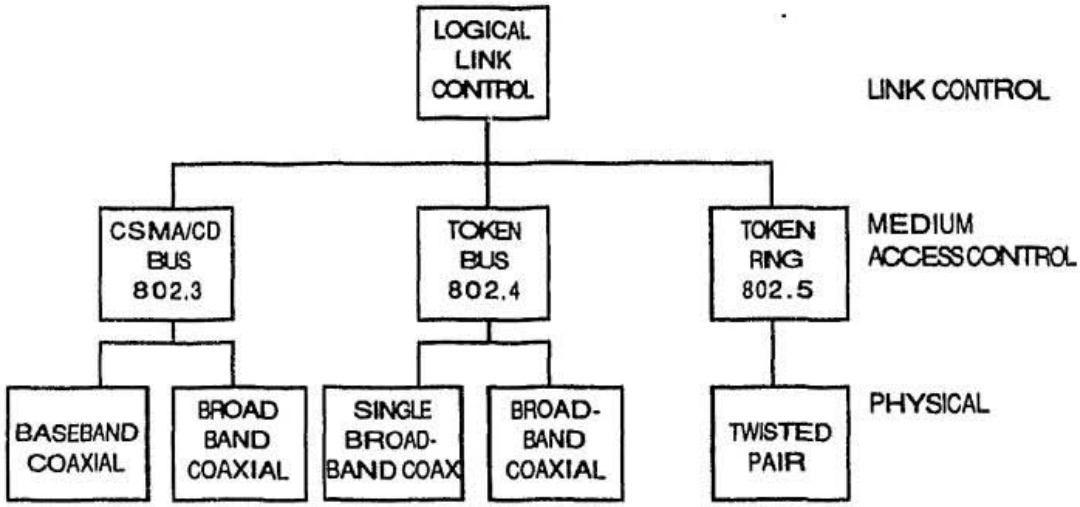

Figure 3.10 The IEEE 802 standard

The logical link control (LLC) layer provides for the exchange of data between service access points (SAPs), which are multiplexed over a single physical connection to the LAN. The LLC provides for both a connectionless, datagram-like service, and a connection-oriented, virtual-circuit-like service.

At the medium access control layer, there are three standards: CSMA/CD, token bus and token ring. Details on the IEEE standards can be found in reference [40].

3.5 SUMMARY

The basic principle of the LAN technology has been introduced. Two major LANs, the CSMA and ring networks have been reviewed. Various LANs including ethernet and token ring, are discussed in this chapter.

All the CSMA networks (including ethernet) are simple and easy to expand since all stations are passively tapped to a common channel. On the other hand, the ring networks provide a collision free transmission. Major drawbacks on the ring network include its lack of expandability and low reliability [12]. If one station fails, the whole network will also go down whereas the CSMA network will continue to operate even when some of the stations fail.

The concept of priority function was also introduced (section 3.3.2.5). The priority function is the major concern in this study for the power substation protection and control.

CHAPTER 4

THE SYSTEM ARCHITECTURE OF THE SUBSTATION PROTECTION AND CONTROL SYSTEM

The application of microprocessors makes it possible to consider a distributed processing method to address the problem of substation protection and control. The advantages of local area networks have already been discussed in the last chapter. Using the modular approach, the total processing requirements of the protection and control system are divided into sub-tasks and allocated to one or more microprocessors. These microprocessors are linked together to form a distributed processing network with a local area network protocol. The distributed processing scheme also provides a flexible and expandable system configuration.

The concept and the structure of the distributed system architecture are discussed in this chapter. The functions of each element in the system are also reviewed. As mentioned earlier in chapter 1, numerous research works have been reported in this area [3-14]. Two of these studies, the Westinghouse's WESPAC system [10,11] and Noor's ASCD protocol [12] are used as a guideline to study the the functionality of the system architecture.

4.1 SYSTEM ARCHITECTURE

To determine the architectures of the substation protection and control system, it is first necessary to consider the criteria of the system. Specifically, the criteria which affect the system architecture are:

- volume of data flow
- response time
- system availability

The data flow of a substation is summarized in figure 4.1. A large amount of data is initiated in the switchyard where measurement devices take instantaneous values of currents and voltages. These analog data must be digitized and transmitted from the switchyard to the control center where processing can take place. The amount of data transfer into the control center is large while the number of control signals sent back to the switchyard is comparatively low. There is also some data flow from the remote central computer which monitors and controls the operations of substation and issues control message back to the control center.

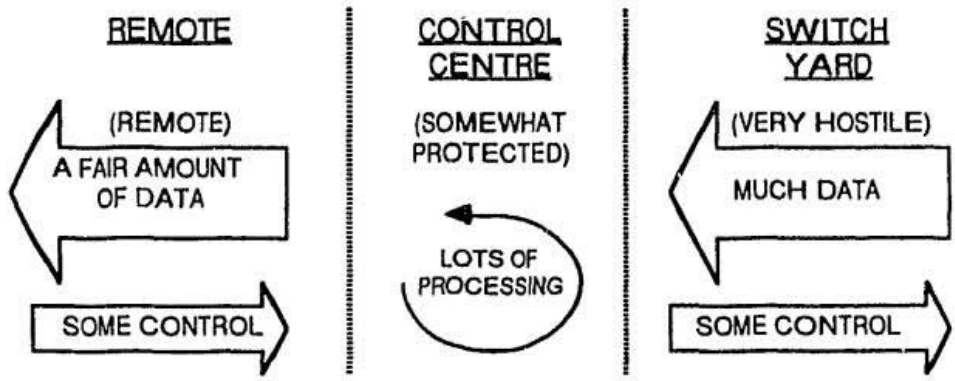

Figure 4.1. Data flow in substation [8]

The characteristics of such distribution of data flow imply a hierarchical architecture as shown in figure 4.2. Beginning at the lowest level, the elements in the hierarchy are:

- data acquisition units (DAU)
- microprocessor cluster
- data communications network
- station computer

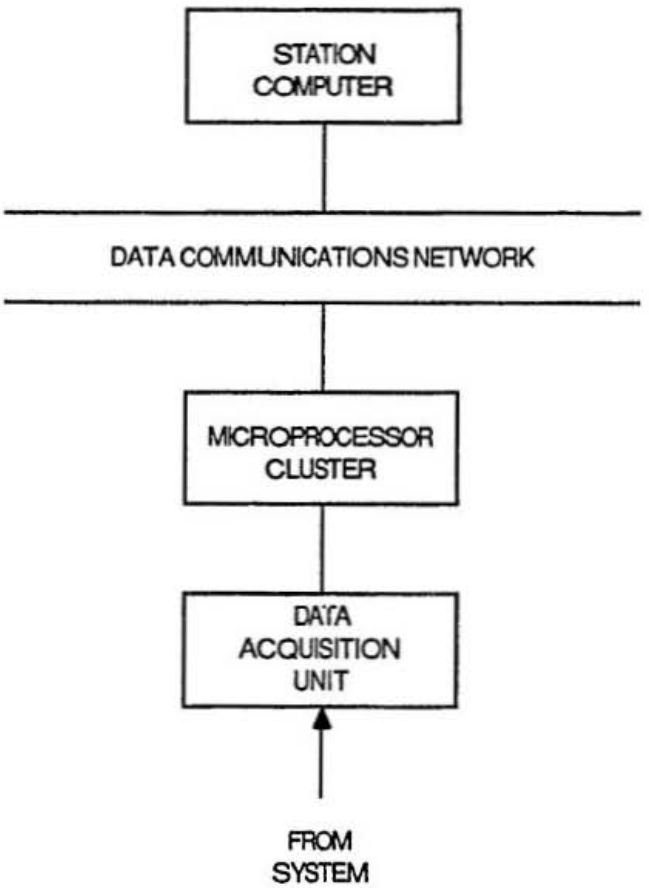

Figure 4.2. The hierarchical architecture of the SPC system

The following describes the function of each element.

4.1.1 DATA ACQUISITION UNITS (DAU)

The data acquistion unit is the interface between the power apparatus and the protection and control system. DAUs are installed throughout the switchyard for the digital protection and control scheme. It takes instantaneous samples of AC voltage and current signals from potential transformers (PT) and current transformers (CT), along with the status of contact inputs such as the breaker state indicator. The analog sample is digitized and sent to the next level of the hierarchy, the microprocessor cluster, for processing the protection and control functions.

In addition, DAUs are also connected to control circuits of the power apparatus for implementing control decisions or requests initiated at higher levels of the hierarchy.

4.1.2 MICROPROCESSOR CLUSTER (MC)

Together with the data communication network, the microprocessor cluster makes up the microcomputer network responsible for protection, control, recording and monitoring functions in real-time. The cluster consists of a number of microprocessors (or microcomputers), memory, and interfaces to the DAUs.

High-speed protective relaying and control programs that reside in the microprocessor, evaluate sequences of incoming instantaneous samples from the DAU

to locate power system faults, monitor and control status of circuit breakers, switches and other apparatus.

In general, the microprocessor cluster performs the following functions:

- initiates sampling requests or control output to the DAU.
- receives instantaneous samples from the DAU.
- executes protection programs to locate fault, and informs the respective control devices to trip the corresponding circuit breakers and alarm the station computer when a fault is detected.
- monitors and controls status of breakers and switches and transmits them to the station computer.
- records instantaneous values of voltages and currents during a fault and transmits them to the station computer for oscillography.
- receives requests for both manual and automatic control operations from station computer or other microprocessors in the cluster, and transmits them to the appropriate DAUs.

4.1.3 DATA COMMUNICATIONS NETWORK

The data communications network is a multi-drop bus local area network which interconnects all the microprocessors and the station computer. It provides data transfer between and within the microprocessor cluster and the station computer. The protocol, topology and the transmission media of the network are selected based on the local area network technology discussed earlier in chapter 3.

4.1.4 STATION COMPUTER

The station computer provides a central data base control point for the entire substation. It collects and stores system information from the microprocessor cluster, via the data communications network.

The station computer consists of a cluster of microprocessors that provide control functions such as maintaining the central system data base and interfacing to the outside world, like the external control devices and the SCADA system.

4.2 THE WESPAC SYSTEM

The WESPAC system is the direct result of a series of projects which were supported by the Electric Power Research Institute (EPRI) and developed by Westinghouse Electric Corporation for an integrated protective and control system in a transmission level substation. The objective of these studies was to employ recent electronics technology to reduce the lifetime costs of substation protection and control functions while achieving performance and feature benefits of new digital approach.

The WESPAC system provides protection, control, and monitoring at the substation. WESPAC, an integrated, modular system, utilizes clusters of multiple microprocessors, multiplexed digital communications, and fiber optic data transmission media in the place of the conventional complement of discrete relays, instruments, and controls with the associated interconnecting wiring.

As mentioned in chapter 1, a number of test trials have been set up by the Public Service Electric and Gas Co. (PSE&G) of New Jersy. These tests have demonstrated the ability of the WESPAC system to withstand the hostile environmental influences of the substation while performing its tasks. Results of these field tests were equal to or better than the conventional system.

The WESPAC system provides some of the protection, control and monitoring functions that have been defined in chapter 2. A brief review of the functional details of WESPAC system provides the basis for design environments in the proposed research of this thesis.

4.2.1 THE SYSTEM ARCHITECTURE

The WESPAC system employs a three-level hierarchy shown in figure 4.3 [10]. The elements of this three-level hierarchical structure are:

- data acquisition units (DAU) with serial data links
- protection cluster (PC)
- serial multidrop data highway
- station computer

4.2.1.1 DATA ACQUISITION UNIT (DAU)

The block diagram of the DAU is shown in figure 4.4. Each DAU contains :

- communications controller
- high-speed analog input interface
- status input interface
- optional electrical to optical (E/O) converter

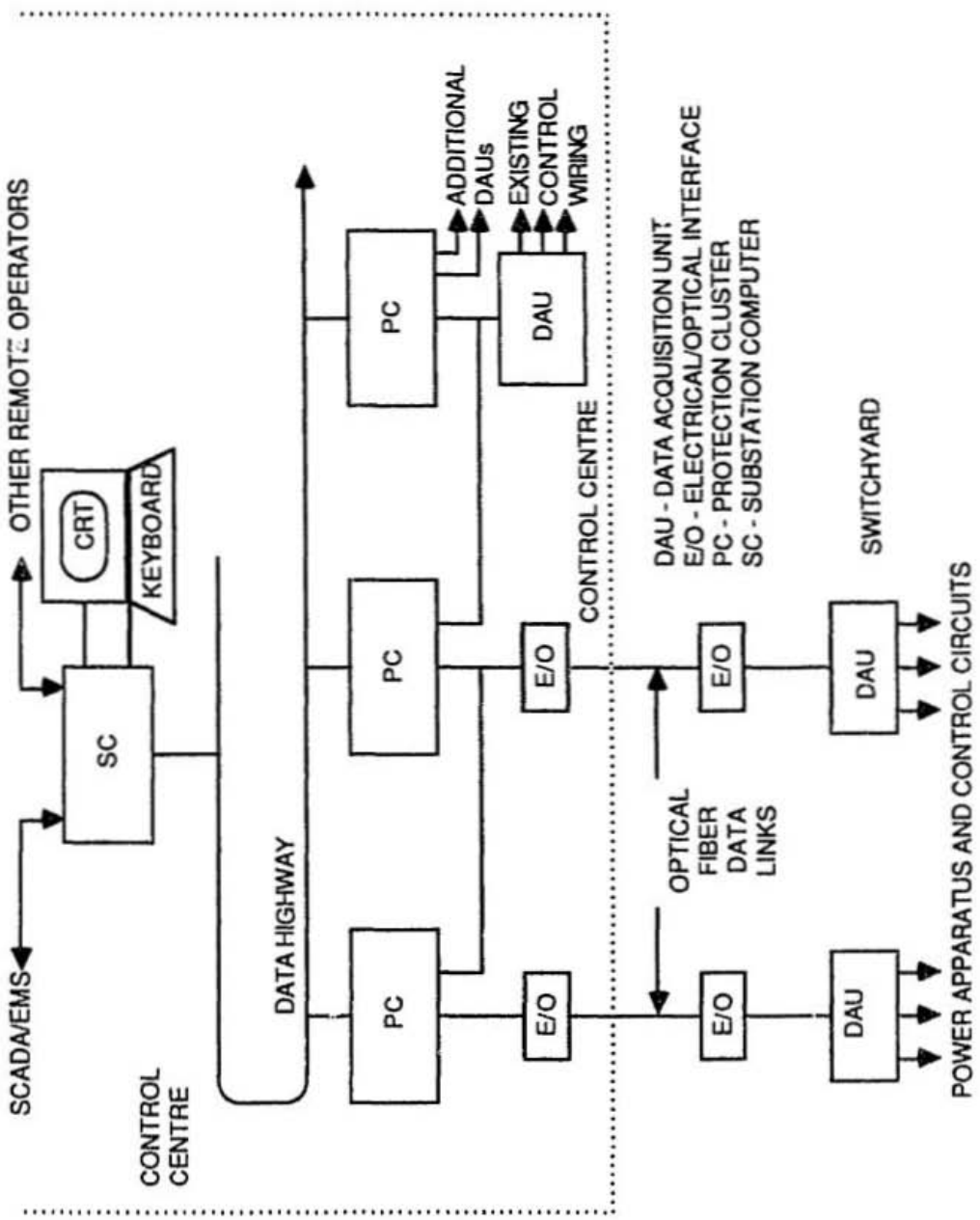

Figure 4.3. The WESPAC system architecture

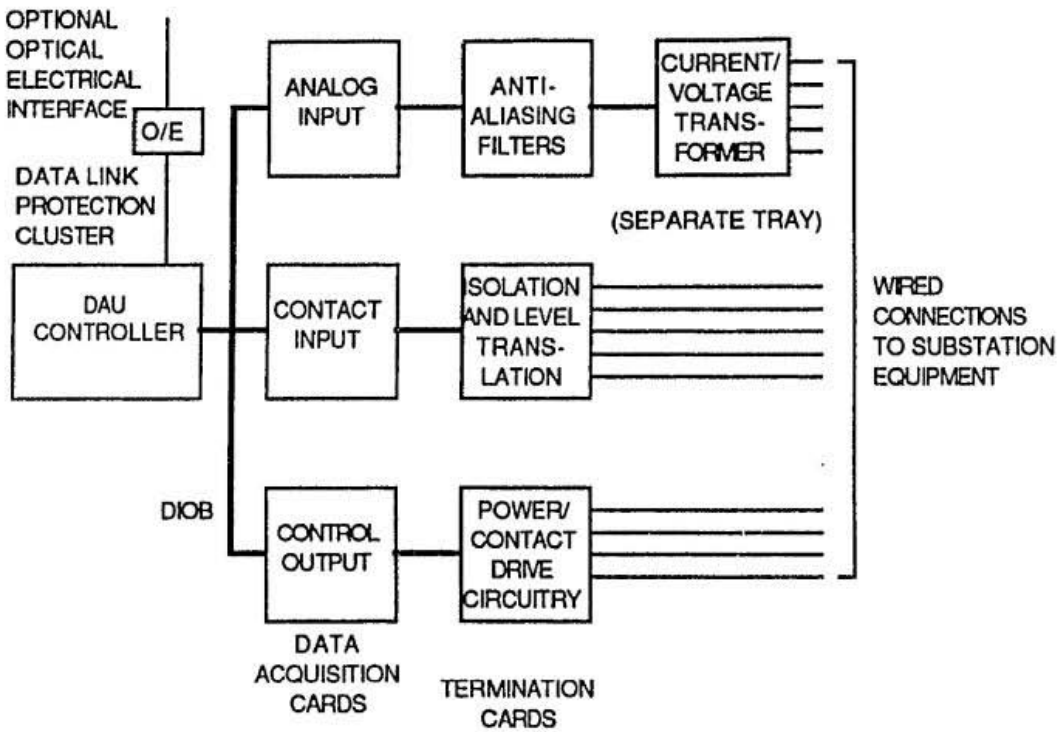

Figure 4.4. The Data Acquistion Unit

The communication controller receives and transmits data over the serial data link and controls the flow of data on the parallel Westinghouse distributed input/output bus (DIOB). When a sychronization message is received from the protection cluster, the controller initiates the sampling and holding of the process for analog inputs, gathers status inputs, converts the analog inputs into digital form, and sends all the information in sequential message form up to the corresponding protection cluster via the serial data link.

Three I/O cards are attached to the DIOB. The high speed analog input (AI) card has sample and hold circuit that can sample a group of six AC inputs simultaneously every 1.04 ms, in synchronism with all other AI cards in all the DAUs of the system. It

also provides anti-aliasing filters, and an A/D converter with dynamic range (16 bits) and a conversion time of 60 μs per point. The contact input (CI) card performs isolation, level shifting and digital contact debounce logic functions. It can recognize status changes in 1 ms, while rejecting contact bounce for up to 32 ms after the closure. The contact output (CO) card provides output relay driving with a high degree of security. It energizes control circuits using SCR's or relay contacts.

4.2.1.2 SERIAL DATA LINKS

The serial data link provides the communications between DAUs and the protection clusters. The transmission medium is either co-axial cable (for DAUs located in the control room) or fiber optic cable (when the DAUs are located in the switchyard). Modems at either end of data link modulate and demodulate the signal using frequency shift keying (FSK) with 3 MHz carrier. Data rate is 1 Mbps. The link network protocol is a subset of IBM synchronized data link control (SDLC) -- ISO standard 3309. When fiber optic medium is used, electrical-to-optical and optical-to-electrical converters are provided.

4.2.1.3 PROTECTION CLUSTER (PC)

The protection cluster consists of clusters of Intel 8086 family of 16 bits microprocessor, memory, and communications controllers as shown in figure 4.5. These elements are connected to a common parallel data bus, the Intel Multibus. Nonvolatile erasable programmable read-only memory (EPROM) holds protection and control programs for each processor, while random access memory (RAM) on each card holds dynamic data. Much of the RAM is shared over the common bus. An additional

nonvolatile memory card holds settings and related data which may be altered in service, yet must be retained when the PC is deenergized. Communciations controllers carry on the data transfer between the PC and the associated DAUs over the serial data links, or among the station computer and other PCs over the multi-drop highway.

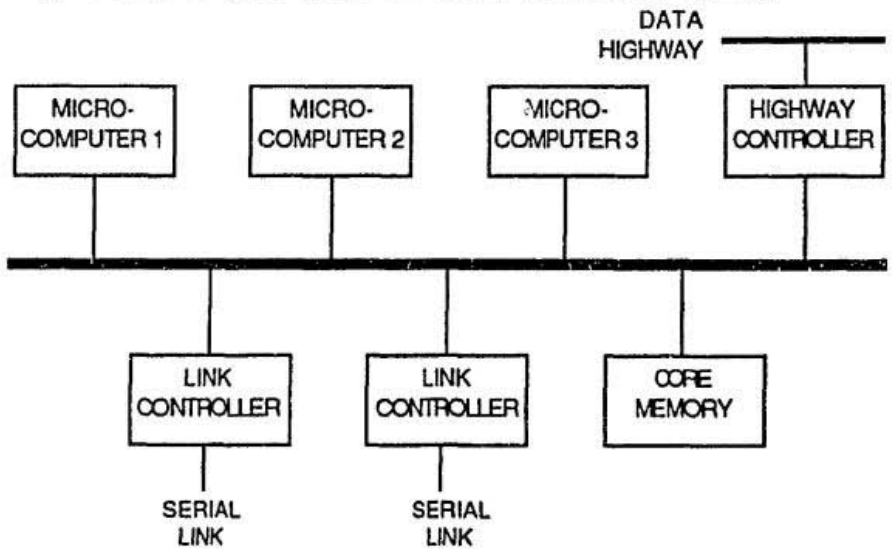

Figure 4.5. The protection cluster

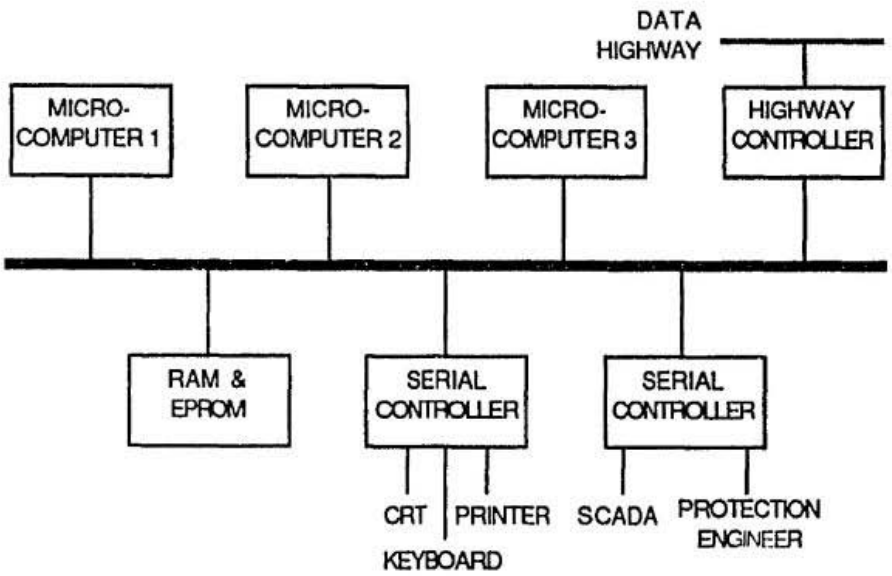

Figure 4.6. The station computer

4.2.1.4 DATA HIGHWAY

The data highway, a multi-drop bus network, provides communications between the protection clusters and station computer. It consists of a co-axial cable modem at each drop. In normal operation, the station computer periodically polls each protection cluster over the data highway. The highway also acts as a communication medium between protection clusters in backup relaying situations. Electrically, the equipment is the same as that used for the serial data links. Since both source and destination are necessary to be specified in a multi-drop network, a subset of the high level data link control (HDLC) - ISO standard 3309 is used as the link network protocol.

4.2.1.5 STATION COMPUTER (SC)

The station computer, like the protection cluster, consists of a cluster of 16 bit microprocessors (see figure 4.6). It maintains the central system data base and provides interfaces to station operators through the man machine interface (MMI) and to system operator and protection engineer through the SCADA system.

At the SC, the MMI consists a 19 inch color graphic CRT, an input keyboard with special purpose and alphanumeric keys, and a printer. The operator can use the keyboard and CRT to display alarm or sequence of events listing, to acknowledge alarms to control circuit breakers and switches, to view oscillographic traces, to check the status of WESPAC system equipment, to change setting, and other functions. Station computer programs and hardware interfaces also make substation data

available to the utility's energy management system and to remote protection or maintenance engineering locations.

4.2.2 SYSTEM OPERATION

Synchronization of the sampling instant in different DAU's is a critical requirement of the WESPAC system. This is achieved by tracking line frequency and dividing each cycle into 16 intervals. At the beginning of each synchronized sampling interval, each protection cluster sends a message to its DAUs through the corresponding serial data links. The DAUs immediately sample and hold analog values. The instantaneous sample values of each AC signal and status input are digitized, formatted, and sent to the protection clusters.

The communication controllers at the clusters deposit these data in predefined shared-memory buffers. When all data have arrived, the protection programs commence processing the new information. The outputs produced by the protection programs are control signals which are sent to the DAUs to initiate breaker operations, plus event messages, oscillographic data collected during fault conditions, and general update of the data base, all of which are sent to the station computer for storage or display.

The station computer receives and stores all this information and in turn sends it, either automatically or on request, to remote operators. It also displays information locally and allows the local operator to initiate manual operations.

4.2.3 SYSTEM HEARTBEAT

A central 960 Hz sampling clock, connected to all of the protection clusters, provides co-ordinating pulses which are communicated over the data link to every DAU. Redundant clocks and clock-pulse signalling paths are provided. Each AC signal and status point in the station is sampled at the same instant. The sampling interval is 1.04 ms which corresponds to exactly 1 sample per 960 Hz power cycle.

The central clock can either operate at a fixed frequency, or can track the station-service power frequency (when available) to maintain exactly 16 samples per cycle regardless of frequency excursions. There is no requirement to maintain any particular phase relationship between the sampling process and the sampled AC waveform. No zero-crossing detection is performed. Each cluster also has an internal timing reference so that timing-clock failure will not disable any relaying or critical control function. The data highway operates at a much slower data rate. There are two categories of interchanges, occurring with intervals of 0.1 sec and 1 sec.

4.2.4 REDUNDACY

The WESPAC system utilizes a cross-linking of DAUs, this permits the sharing of DAU data among PC's for several zones. In addition, each PC in the WESPAC system can execute its relaying tasks without the data highway connected or operational - protection for multiple zones is not dependent on a single hardware element.

4.3 NOOR'S ASCD NETWORK

The ASCD protocol is the result of a graduate research done by Asgar Ibn Noor at the University of Calgary in 1982. The objective of his research was to design a microcomputer network to meet the processing requirement of a real-time application, the substation protection and control system.

In his work, a number of network protocols were considered, ALOHA, CSMA, TDMA and ASCD. Simulation results showed that the ASCD protocol has the best overall performance.

Like the WESPAC system, Noor also employs a three-level hierarchical structure as the system architecture. Most of the elements perform the same functions. The major difference is the network access scheme, Noor proposes to use an assigned slot CSMA/CD (ASCD) protocol whereas a subset of high level data link control (HDLC) protocol - ISO standard 3309 is used in the WESPAC system.

4.3.1 ASSIGNED-SLOT-CSMA/CD PROTOCOL

The ASCD protocol is designed to improve the channel utilization (i.e. the network performance). This scheme gives priority to packets transmitted by certain nodes (stations). Under this scheme, several nodes are allowed to share a common slot. In such an algorithm, there exists a tradeoff between time wasted in collision and control overhead.

In ASCD protocol, time is divided into frames, each frame containing an equal number of L slots. Each of the M = N/L users is assigned to a slot where N is the total number of nodes. The users monitor their own transmission and detect a collision if it fails to detect its own transmission. The collision can be detected within a one-bit period.

The users are grouped in accordance with their physical distance of separation. This is done to avoid the propagation delay. Under this scheme, a higher priority subset of users may get more than one consecutive slot. If a user detects too many collisions then it can move up to the next higher order slot; it moves back to its original assigned slot as soon as its packet is successfully transmitted.

In the ASCD protocol, a node senses the channel and operates as follows:

1) if the channel is idle, start transmission at the beginning of the next assigned slot.

2) if a collision is detected during transmission, abort the transmission immediately.

3) if the channel is busy, defer transmission for a random amount of time and execute the algorithm all over.

4) if 64 consecutive collisions are detected then switch to the next higher slot and go back to step 1.

If users belonging to the same slot start transmission simultaneously, only one can obtain the channel mastership. This channel mastership is illustrated in figure 4.7.

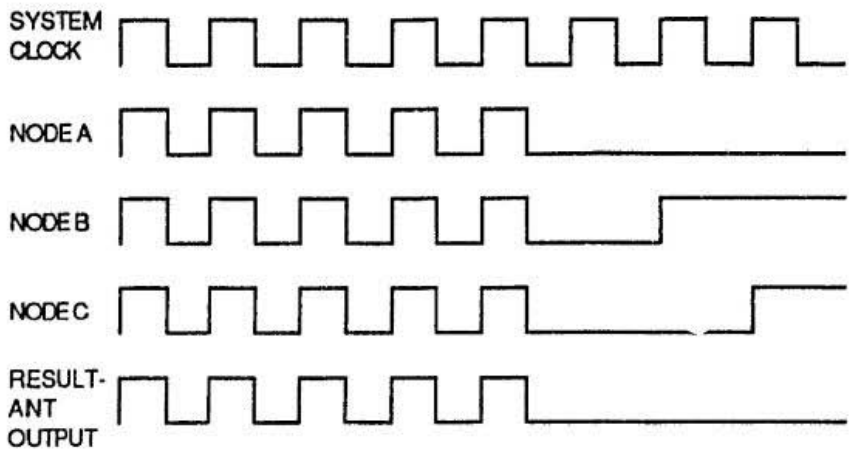

Figure 4.7. Collision detection machanism for the nodes assigned to the same slot

Node A, node B and node C are the members of the same slot and start to transmit at the same time. Since their transmissions are identical during the first 6 clock pulses, they do not detect any collision. On clock pulse 7, node A and node C transmit 0 and node B detects a collision because it transmits a 1 and receives a 0. As a result, it ceases transmission. On clock pulse 8, node C detects a collision because it transmits a 1 but receives a 0. As a result, it ceases transmission. Node A continues its transmission.

Under this scheme, only one member of a slot transmits at any time, the rest reschedule their transmission to a future frame. This technique guarantees that the high priority packet always gets through the network.

The above algorithm works, if the separation between the farthest two nodes in a slot is less than half the wavelength of the operating speed of the channel. If this distance criterion can not be met, then none of the nodes gains channel mastership during contention. The contending nodes then wait for a period calculated from a

negative exponential series and retry for the channel. By optimizing the number of users (M) assigned to each slot, the delay-throughput performance can be improved substantially. Here, collisions are bound to occur, but the average delay is reduced, because of the dynamic nature of the algorithm.

One additional disadvantage of this ASCD scheme is the excessive overhead of the time slot. If only one time slot has a message to send in one time frame, L-1 slots of time are wasted. This wasted time is substantial over a long period of time. The condition is worse when many data are queueing up one particular time slot. These limitations of the ASCD scheme led to the development of the proposed network protocol in this study. The proposed protocol still contains the message priority function, but reduces the time slot overhead substantially, thereby improving the overall network performance.

4.4 SUMMARY

The hierarchical structure of the substation protection and control system has been described including the function of all the elements.

Two examples were given to illustrate this hierarchical structure, namely, the WESPAC system and Noor's ASCD network. From these discussions, the need for a more efficient network protocol became obvious. The detailed design aspects of the proposed protocol are described in the next chapter.

CHAPTER 5

CONCEPTUAL DESIGN OF THE PROPOSED NETWORK

The architecture of the substation protection and control system was outlined in the previous chapter. Two examples, the WESPAC system and Noor's ASCD network, were used as an aid to demonstrate the functions and the architecture of each element in the hierarchical structure. In this chapter, a new design approach is proposed by employing recent nigh speed Local Area Network (LAN) technology.

The LAN technology has been introduced in chapter 3. This background information is used to develop a new design approach best suited for the application of substation protection and control.

5.1 NETWORK REQUIREMENTS

The hierarchical structure outlined in the last chapter indicates that the volume of data flow in each level of the hierarchy is the major concern of the system. In addition, response time and availability of the system are also vital in the design.

5.1.1 VOLUME OF DATA FLOW

The WESPAC system utilizes three different networks to provide the data transfer of all the protection and control functions. The first network is the serial data link which provides the data transfer between the DAU and the protection cluster. This network is implemented with a subset of IBM synchronous data link control (SDLC) network protocol. Three types of data are mainly involved in the network, the sample data (from DAU to protection cluster), the control data (from protection cluster to DAU) such as synchronization message, and the trip signal (from protection cluster to circuit breaker via the DAU). A lot of data transfer is occurring at this level, mainly the flow of sample data and the control data. They are flowing at a frequency of 960 Hz (i.e. once every 1042 μs). Trip signals enter the network only when faults are located.

The second network is the multibus system within the protection cluster. The main function of this network is to provide a communication medium among the protection microprocessors (digital relays) and the cluster shared memory. The WESPAC system uses the Intel Multibus system in this level. A fair amount of data transfer is involved at this multibus network, mainly the sample data which flow within the protection cluster, and data from the protection microprocessors (digital relays) to and from the common shared memory.

Finally, the last network is the multi-drop data communiation network which interconnects all the protection clusters and the station computer. The data communication network is normally used in the master-slave mode with the station computer acting as the master and periodically polling each protection cluster for current system status information. In the WESPAC system, this data highway is implemented with a subset of high level data link control (HDLC) network protocol while

Noor proposed an assigned-slot-CSMA-CD (ASCD) for this application. The volume of data flow at this level is considerably low; most of these data are for control functions and current system status information. Some additional data transfers for backup relaying are also involved.

In summary, the data involved in a substation protection and control system can be grouped into five categories.

1) Sample data : This contains all the sample data sent from DAUs to the protection cluster via the serial data link, they are sampling at a rate of 960 Hz.

2) System data : The system data includes data which is running within the protection cluster (i.e. the multibus system), in the data communication network for current system status information, and some other event data.

3) Control data : This includes all the control commands (messages) for the system control functions.

4) Fault data : The fault data are the trip signals which are sent to switch off the circuit breakers from the protection microprocessors via the DAU.

5) User data : This includes the manual control commands and system information which are requested by the operator from station computer and dispatch center.

Furthermore, with the exception of the sample data, all the communications of these data can be provided by a totally distributed local area network. This totally distributed LAN is discussed in section 5.2.

5.1.2 RESPONSE TIME

The operating time of conventional relays usually does not exceed 50 ms, that is 3 cycles on a 60 Hz power source [9,23]. This 50 ms operating time spans from the start of the sampling of the power signal till the time that the breaker is disconnected after a fault is located. This 50 ms operating time is considered as the maximum time limit for the substation protection system.

For transformer protection, all digital protection algorithms can be executed in less than 0.55 ms [17]. These execution times have been obtained based on an 8086 16-bit microprocessor and the Intel 8087 numeric data co-processor with a clock frequency of 8 MHz. A trip time of 15 ms was obtained. These tests were performed at the Memorial University for transformer protection.

For transmission lines protection, most relay algorithms have a trip time of less than half a cycle, that is 8 to 9 ms [25]. With the exception of a modified Fourier technique in a 19 km of a 500V transmission line which has an average real-time response time of 21 to 26 ms for in-zone faults. These results were obtained from a Motorola 6800, 8 bit microprocessor based digital relay. Better results are expected if the Motorola 6800 is replaced with a 16 bit microprocessor such as the Intel 8086 microprocessor.

These results show that most digital relays have a trip time of less than one cycle of 60 Hz power source, that is,.16.6 ms. This trip time is well within the 50 ms maximum time limit. This time is obtained from a relay system that the digital relay is connected to the DAU with a serial data link, and the trip signal is sent to the DAU from the protection microprocessor. This trip signal is then redirected to the circuit breaker to switch the circuit off. This system has the protection microprocessor connected to the respective

circuit breakers directly. Therefore a 100% system availability is ensured by the system provided there is no hardware damage.

5.1.3 SYSTEM AVAILABILITY

System availability plays a vital role in network performance. Availability is affected by the system architecture and hardware reliability. Most of the direct link (point-to-point link) has 100% system availability, like the serial data link. As a result, a trip signal is guaranteed to be sent to the circuit breaker as soon as a fault is located, provided there is no hardware damage.

For other network links (not point-to-point link), users share all the resources of the network (including the channel). Some contention schemes must therefore be provided to access the channel. Due to the contention, message will be delayed until its owner gains access to the channel, therefore the system availability is no longer 100%. The contention delay is dependent mainly on the performance of the contention scheme (i.e. network protocol).

Furthermore, hardware reliability can be improved by a constant monitoring and self-checking scheme. Appropriate redundancy on the system can also enhance the reliability of the hardware.

5.2 THE PROPOSED NETWORK

A totally distributed local area network is proposed for the real time application of substation protection and control. The configuration of this totally distributed LAN is illustrated in figure 5.1.

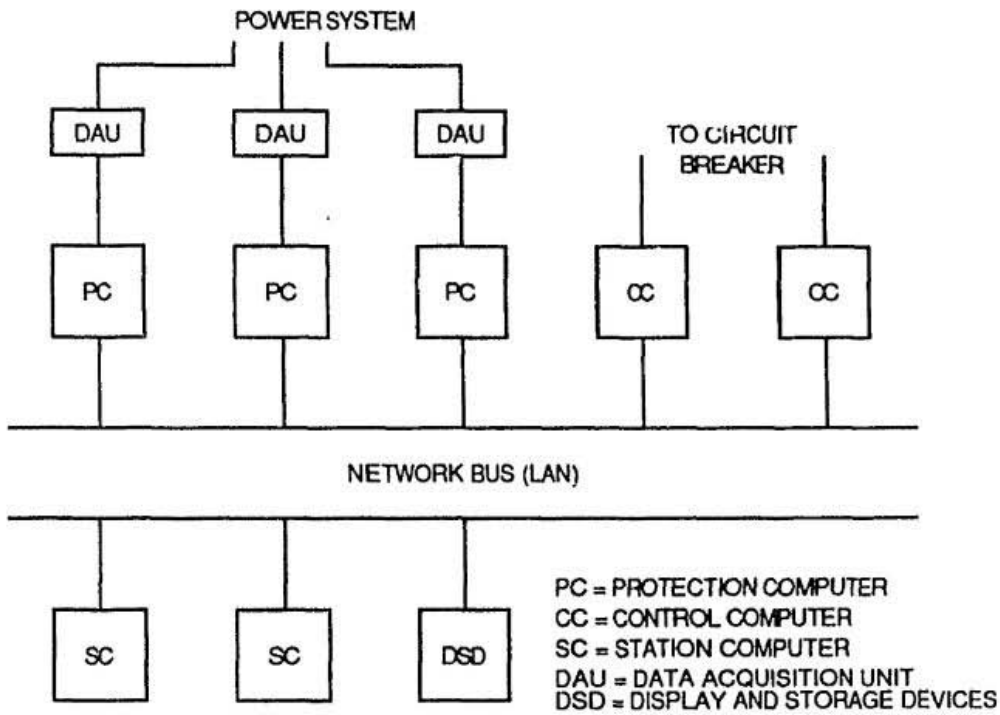

Figure 5.1. The configuration of a distributed LAN for substation protection & control

The function of each component is outlined below:

Data Acquisition Unit (DAU) : To sample AC power signals (currents by current transducers and voltages by voltage transducers) and to convert them into digital form for protection processing.

Protection Computer (PC) : The protection computer determines the condition of the power apparatus. Once a fault occurs, a fault message is sent to the corresponding control computers to trip the corresponding circuit breaker via the LAN. Each protection computer provides a particular protection function. For example, transformer protection uses the Walsh protective algorithm [17]. The communications between the protection computers or/and other computers are via the LAN.

Control Computer (CC) : The control computer provides the automatic control functions as well as the tripping of the circuit breaker. To trip a breaker, the control computer must receive a fault message from the protection computer via the LAN. Each control computer can control several circuit breakers. Depending on where the fault message came from, the program within the control computer determines which breaker should be turned off to isolate the faulty equipment accordingly. The control computer can also be a station computer or a gateway which is connected to the outside world.

Interface (I) : The interface provides the communcations between the control computers and the power system. It makes the necessary data conversion to ensure the data between the control computers and power system are compatible.

Printer, Memory and Display devices : These devices provide the monitoring and recording functions of the substation protection and control.

LAN : The LAN provides the communications for all the data transfer within the substation except for the sample data. The transfer of sample data are performed by the direct serial link between the DAUs and protection computers.

As discussed in the last section, when a multiple access network is running, system availability is no longer 100% due to the contention. Therefore, it is interesting to examine if the proposed network can fullfil the response time requirement.

Of all the data, only the trip data (fault message) has a vital maximum time of 50 ms. Some control data also have a time limitation, however, it is not considered as important as the fault message. Since most of the digital relays have a trip time of less than one cycle [17], that is well within the 50 ms limit.

For the totally distributed system, it is assumed that the execution time of 1 ms for all of the relay algorithms (in test trial, most algorithms can be executed in less than 0.55 ms [17,26]). It is further assumed that the fault message can be coded into a 256 bits packet (including all the control, address, header message as well as the fault message).

If the data rate is 1 Mbps, the transmission time for the fault packet is therefore 0.256 ms provided there is no message delay (contention). Assuming propagation delay time is negligible on a 1 km coaxial cable LAN, the end-to-end propagation delay is 5 μs [39,40].

To determine the allowable range of message delay, it is further assumed that the operating time from start of the sampling until the protection computer receives the

sample data and the operating time for the control computer to determine which breaker should be turned off until the breaker is actually tripped is 8 ms each. Therefore

trip time = 8 ms + 1 ms + 0.256 ms + 8 ms

= 17.256 ms

allowable range of message delay = 50 ms - 17.256 ms

= 32.744 ms

With a data rate of 1 Mbits/s, a total of 33 Kbits data can be transmitted in 33 ms, eg. approximately one hundred of 256 bits data, that is a lot of data transferring. This calculation indicates that a totally distributed network as shown in figure 5.1 may be able to satisfy the requirements of the substation protection and control system.

To ensure the fault message has a minimum message delay time, a message priority LAN protocol is employed. This message priority scheme makes sure that the fault message has access to the channel as soon as the channel is available (ie. the current transmission is completed).

However, the above calculations are based upon a number of assumptions. To verify the network performance, a performance analysis of this network is required. A simulation of the message priority LAN is performed in next chapter.

5.2.1 THE ADVANTAGES OF THE NEW APPROACH

The new design approach of totally distributed LAN has the following advantages:

5.2.1.1 SIMPLICITY

Only a single local area network is required in the totally distributed LAN, while the other systems required two different networks, the multibus and the data communication networks. This simple architecture improves the reliability of the whole system.

Furthermore, each protection computer, control computer and other devices are all on a one-to-one basis, additional system simplicity is therefore achieved.

5.2.1.2 MODULARITY

The totally distributed LAN ensures a completely modular design. Each computer has only one specific function to perform, such as Walsh algorithm protection on transformer, to trip the particular circuit breakers, to isolate the fault, and to display the current system information etc. This completely modular approach eases and simplifies the whole system design.

5.2.1.3 EXPANDABILITY

The modular design accommodates modification or growth of the substation with minimal new wiring and engineering design. The only change is to add the corresponding address to the network when an additional unit is put on the system.

The fact that the system is integrated also provides flexibility in system performance evaluation because changes in the system can be easily located and tested against all the other functions which can be affected by these changes.

5.2.1.4 MESSAGE PRIORITY

The message priority scheme improves the overall network throughput. The main requirement is to isolate the fault in less than 50 ms once a fault is located. Since the fault message has the highest priority, it is always ready to transmit with minimum message delay time. The message delay for other data is also designed to be minimal in this message priority protocol. Details of this message priority protocol are described in section 5.3.

5.2.2 PRIORITY CLASSES

As summarized in section 5.1, there are four kinds of data involved in the proposed network:

-- fault data

-- system data

-- control data

-- user data

Depending on their importance and their frequency, these data are classified into three priority levels:

1) Fault data : The fault data have the highest priority for obvious reasons. This data is transmitted in such a way that the total trip time is less than 50 ms.

2) System data : The frequency of the data in this class is the highest. This data rate is dependent on the number of devices in the system and how often the signals are displayed or stored. For a system with N = 100 devices, and one sample information is stored and recorded on the average in one power cycle (60 Hz), at least 100 sample data are therefore put into the network in 16.6 ms, for a packet size of 256 bits, a data rate of approximately 1.5 Mbps is necessary.

3) Control and User data : The control data and user data are combined into one priority. Some of the control data have time requirements, however, these time limits are considered not significant. Most of the user data are for information gathering, they are not especially important.

5.3 THE MESSAGE PRIORITY CARRIER SENSE MULTIPLE ACCESS PROTOCOL

The message priority CSMA/CD (MPCD) protocol has been proposed for efficient channel utilization. This scheme assigns priority to the data (message) based on its type. Three priority classes are selected for the substation application as described in section 5.2.2.

To implement priority functions in a LAN, three basic problems must be considered:

1) to identify the instants at which to access the current highest priority with ready message.

2) to design a mechanism by which to access the highest nonempty class.

3) to design a mechanism which assigns the channel to the various ready users within a class.

In the proposed MPCD protocol, the first two problems are solved by means of reservation bursts and carrier sensing [41], and the third by employing CSMA/CD and the collision detection mechanism developed by Noor in figure 4.7.

The CSMA protocol has been introduced in chapter 3. The MPCD protocol is described in detail as follows. This protocol is implemented in a common bus topology.

5.3.1 THE MPCD PROTOCOL

Like any other CSMA schemes, users in the MPCD protocol monitor the activity on the channel at all times. The assessment of which ready message has the highest priority class is done at the end of each transmission period, i.e. every time the carrier on the channel goes idle. The end of each transmission period is signalled by an end of carrier (EOC) packet. When detected at user, EOC establishes a time reference for that user.

Following EOC, the channel time is divided into slots with the size of a slot (referred to as reservation-slot) equal to $(2\tau+\gamma)$, where γ = the shortest burst time which can be reliably detected for a transmission and τ = end-to-end propagation delay of the LAN.

For each user, messages are ordered according to their priority. The priority of a user at anytime is the highest priority class of messages present in its queue.

Let i denote an arbitrary user, and $T_e(i)$ denote the time of end of carrier at user i. Let P(i) denote the priority level of user i at time $T_e(i)$. The flow chart of the MPCD protocol for data (i) is shown in figure 5.2 and it performs the following algorithm.

1) If, following $T_e(i)$, the carrier is detected in reservation slot h, with $h < P(i)$ (i.e. some user has priority h higher than P(i) and access right must be granted to class h), then user i awaits the next end of the carrier EOC (at the end of next transmission period) at which time it re-evaluates its priority and repeats the algorithm.

2) If no carrier is detected prior to the jth reservation slot, with $j = P(i)$, then user i transmits a short burst of unmodulated carrier of duration γ at the beginning of reservation slot j (therefore reserving channel access to priority class P(i)). Since there may be other ready messages which have the same priority of P(i), immediately following this reservation slot, the network operates according to CSMA/CD protocol (see chapter 3). That is, it senses the channel and proceeds as follows.

2a) If the channel is sensed idle, it transmits the message. As long as no collision is detected, transmission of the message proceeds. If a collision is detected, transmission is aborted and the user reschedules the transmission of the message to some random time and repeat the CSMA/CD algorithm.

2b) if the channel is sensed busy, then the user awaits the next EOC and re-evaluates its priority level and repeat the entire algorithm.

2c) if, during the time that the channel access is granted to class P(i), some user i' generates a new message of the same priority, then i' transmits the new message provided that the channel is sensed idle as (2a), and proceeds as (2b) if the channel is sensed busy.

3) if, following EOC, no reservation burst is detected for L consecutive reservation-slots, where L is the total number of priority classes available in the system, then the channel becomes free to be accessed by all users regardless of their priority, until a new EOC is detected.

Therefore, by means of short burst reservations following EOC, the highest priority class (nonempty) is granted exclusive access right, and messages within that class can access the channel according to the CSMA/CD protocol.

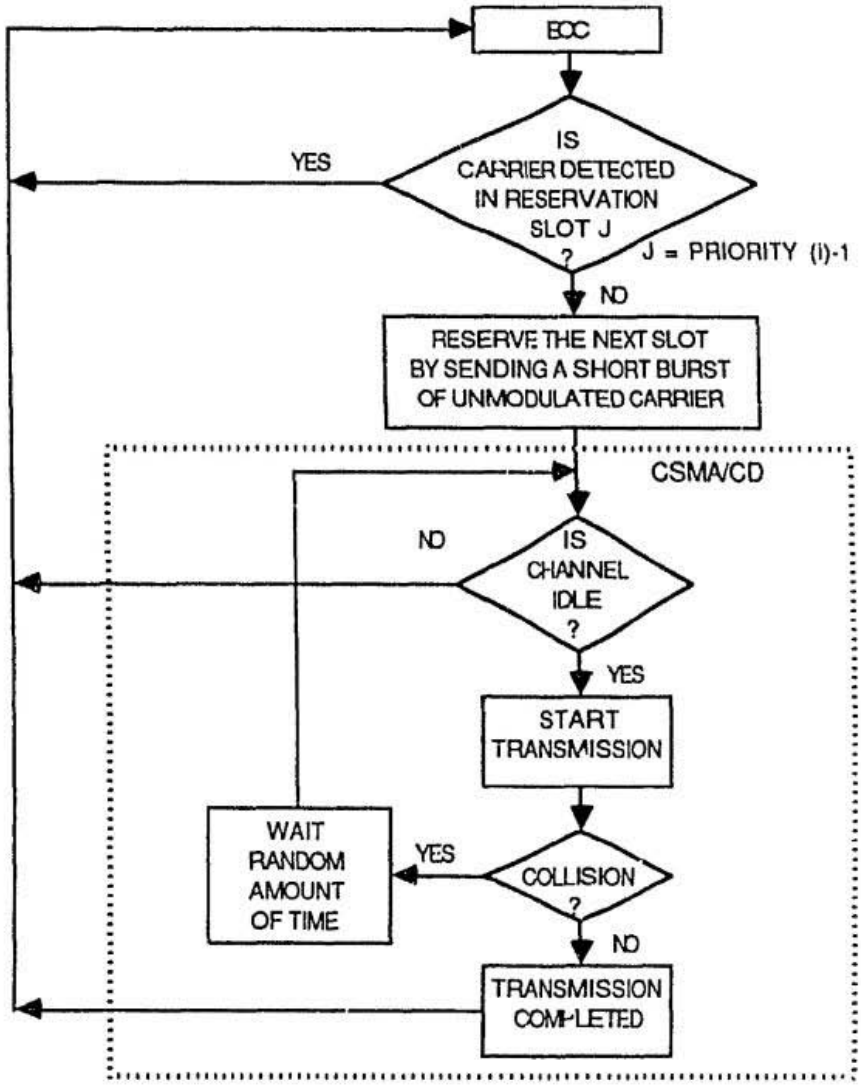

Figure 5.2. The flow chart of the MPCD protocol for data (i).

Note that the above algorithm corresponds to a nonpreemptive discipline, that is an user which has been denied access does not re-evaluate its priority until the next EOC. However, by accessing the highest priority at the end of each transmission period, whether successful or not, the scheme allows higher priority messages to regain the access right without incurring substantial delays.

A distinct advantage of the MPCD protocol compared to Noor's ASCD protocol is its variable reservation period. In MPCD, the maximum reservation period is equal to L slots, since each slot has a time of $2\tau+\gamma$, therefore, maximum reservation period is equal to $L(2\tau+\gamma)$. However, this reservation period is variable; it depends on the current priority level. If the current priority level is i, the overhead of reservation period is equal to $i(2\tau+\gamma)$. For example, if the current priority is the highest, only a $2\tau+\gamma$ overhead time is used for reservation, whereas, in Noor's ASCD, a fixed reservation period is used. As a result, the MPCD saves a substantial overhead time for reservation, especially for users having large amount of high priority data. One should note that the overhead incurred in a reservation period following EOC is a function of the current highest priority level. The higher this class is, the smaller the overhead and the delay to gain access right.

The numerical results of the simulation of this MPCD protocol are presented in chapter 6.

5.4 SUMMARY

A new design approach was developed for substation protection and control. This new design approach utilizes a totally distributed LAN with a message

priority (MPCD) protocol. The network requirements were established. The most important requirement is that the system must have a trip time of less than 50 ms.

The substation data were classified into three priorities. Starting with the highest priority, they are: the fault data, the system data, and the control data & the user data.

A detailed description of the MPCD protocol was given. Initial calculations indicate that the totally distributed MPCD LAN can meet the substation protection and control requirement. A performance analysis and a simulation for the MPCD LAN are performed in the next chapter.

CHAPTER 6

PERFORMANCE ANALYSIS

The objective of this analysis is to predict the performance of the MPCD protocol. This performance analysis calls for a model capable of predicting the channel utilization and the average message delay time of the proposed protocol. In addition, the model should be structured in such a way that its performance can be compared to Noor's results.

In general, two approaches can be used to evaluate the performance of a computer network: by applying the queueing theory and by computer simulation.

Queueing theory [41,42,43] provides a powerful analytic technique for the analysis of a computer communications network. Essentially in a queue, customers (packets) arrive at some service facility (node,channel) with requests for service and leave the facility after obtaining service. In general, queueing theory deals with quantities such as the number of customers at the facility at any time, the total amount of time required to process individual customers through the facility and utilization rate of the facility. Several variations of the CSMA protocol have been analyzed by Tobagi and Kleinrock [44] using the queueing theory. They modeled the system using an M/M/1 queue with a buffer capacity of one. Lam [45] analyzed a CSMA protocol using an M/G/1 queue model with a buffer capacity of one. Noor [12] used a Markovian model for studying the behaviour of the ASCD protocol. His model consists of two Markov chains, one chain describing the state of the node, referred to as the user Markov chain,

and the other describing the state of the network, referred to as the network Markov chain.

The queueing theory approach is limited by the complication of the mathematics involved. It is therefore usually used only when analyzing a small network. Numerous assumptions are needed to be made in order to solve the complicated mathematical equations, therefore, reducing the accuracy of the results when the analysis is used on a large size network as in power substation environment.

Simulation is an efficient and a widely used technique to examine the behaviour of a computer network system. In a simulation [46,47,48], a computer is used to evaluate the model numerically over a time period of interest, and data are collected to estimate the desired characteristics of the model.

In this study, the simulation approach is taken to analyze the performance of the proposed protocol. This chapter describes the simulation modeling of the MPCD protocol. The simulation results of various traffic loads and operating conditions are then used to perdict the performance of the network.

6.1 CONSTRUCTION OF THE MPCD SIMULATION MODEL

A computer network is a discrete event system since the state variables, e.g. the number of ready data in the network, change only when a packet (data) arrives or when trasmission of a packet is completed [49]. Discrete event simulation concerns the modeling of a system as it evolves over time by a representation in which the state

variables change only at a countable number of points in time. These points in time are the ones which may change the state of a system.

Due to the dynamic nature of the discrete event simulation, special techniques are required to advance the simulation clock and update the event list queue.

6.1.1 SIMULATION CLOCK

Two principal approaches are available to advancing the simulation clock, namely, next-event time advance and fixed-increment time advance.

With the next-event time advance approach, the simulation clock is intialized to zero and the times of occurrence of future events are determined. The simulation clock is then advanced to the next (first) of these future events, at which point the state of the system is updated. This process of advancing the simulation clock from one event to next event is continued until some prespecified terminating condition is satisfied. Since all state changes occur only at event times, periods of interactivity in a system are skipped over by jumping the clock from event time to event time.

With fixed-increments time advance approach, the simulation clock is advanced in increments of exactly Δt units for some appropriate choice of Δt. After each update of the simulation clock, a check is made to determine if any events were scheduled to occur in this interval, these events are then considered to occur at the end of the interval and the system states are updated accordingly.

For the MPCD protocol, due to the priority scheme (reservation period), a combination of both these two approaches is employed to implement the simulation clock. First, the next-event time advance technique is used to determine the next event time (an arrival or a departure event) The simulation clock is then advanced to this time to update the system state variables At this point, the simulation is switched to the fixed-increments approach, the increment time Δt is set equivalent to the reservation period to determine which ready data within that period of time has the priority to transmit first if there are ready data waiting on the event list queue Once a transmission is started, the simulation is switched back to the next-event time advance technique to look for the next event time

6.1.2 EVENT QUEUE DISCIPLINE

The event queue discipline concerns the behaviour of events in the waiting line For the MPCD network protocol, data with the highest priority in the queue will transmit first. If more than one piece of data has the same priority (highest), they will contend to determine which data will go first and the others will return to the event queue. The contention mechanism is CSMA/CD as described in section 3.3.2.4. The arriving data will be put into the event queue when the current transmission is not completed. If no data is on the queue, simulation clock is set to the time of the next scheduled arriving data.

6.2 RANDOM NUMBER GENERATION

In the simulation, a random number generator is required to generate the data traffic distribution. A random variable is defined to be a function satisfying certain conditions in probability theory. Since computer systems are designed to be deterministic with respect to individual programs, it is not possible for a program to have truly random behaviour. In this study, a prime modulus multiplicative linear congruential generator (PMMLCG) is used to generate the random number for the traffic distributions. The LCG model, a Possion distribution, was first introduced by Lehmer [49]. The formula is given as by

$$Z_i = (aZ_{i-1}+c)(\text{MOD } m)$$

where m = modulus,

a = multiplier,

c = increment,

Z_i = random number,

Z_0 = random seed

The PMMLCGs pseudo random number generator can be given as [49]

$$Z_I = (75Z_{i-1})(\text{MOD } 2^{31}-1)$$

6.3 SIMULATION MODEL

The source code of this MPCD simulation model is included in the appendix A. The program is written in C programming language and in a SMPL simulation language [50]. The SMPL simulation environment is also listed in appendix B. This program is also based on the following assumptions [39]:

- the arrival process of packets (all priority) is Poisson distributed.
- the packets are distributed to each source node uniformly.
- the regenerate process is exponentially distributed.
- the channel is assumed to be noiseless.
- a node may, at one time, be either transmitting or receiving, but not both. However, the node can switch from one mode to the other without any delay.

6.3.1 PACKET SIZE

The packet size determines the service time of the data in the simulation. A packet not only contains the data field, it also contains source and destination addresses and other control information. Figure 6.1 shows the IEEE 802 CSMA/CD packet structure [40]. The individual fields are defined as follows.

1. Preamble : A 7 byte data used for bit synchronization
2. Start frame delimiter (SFD) : Indicates the start of the frame.
3. Destination address (DA) : Specifies the station(s) for which the frame is inteded.
4. Source address (SA) : Specifies the station that sent the frame.
5. Length : The length (in byte) of the data field.
6. Data : Data field.
7. Pad : A sequence of data added to ensure that the frame is long enough for CSMA/CD operation.
8. Frame check sequence (FCS) : Redundancy check sequence.

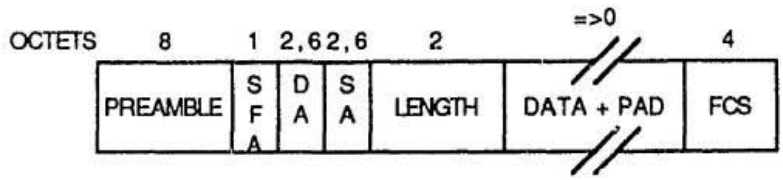

Figure 6.1 CSMA/CD frame format.

Beside the data field, an IEEE 802 CSMA/CD packet varies from 152 to 216 bits. In the simulation, a packet size between 64 bits to 1024 bits is used. Packets of any size in this range can be generated at any time during the simulation.

6.3.2 DATA PRIORITY LEVEL

All data are generated with a Poisson distribution [49]. The mean arrival rate of the Poisson distribution is specified during the simulation. The priority level of each data is assigned using the Monte Carlo method [41].

6.3.3 PERFORMANCE STATISTICS

All statistics are based on the performance of individual nodes. Data are collected on the following quantities: the mean delay time, the average queue time, the mean service time and the channel utilization. In addition, the response time of the fault data is also recorded.

6.3.4 SIMULATION RESULTS

Several observations are made about the MPCD protocol. The channel utilization versus average arrival rate for various fixed packet sizes is shown in figure 6.2 (Channel utilization is defined as the total data transmission time divided by the total time). The graph shows that the channel utilization of the MPCD protocol is stable even at high traffic load. The utilization does not decrease in high load. The upper limit of the channel utilization is about 85%, whereas in 1-persistent CSMA protocol is 51% [39].

The relationship between the average delay and average arrival rate for various fixed packet size is plotted in figure 6.3. The curves indicate that the average delay depends on the average arrival rate and the packet size. The rate of change of delay is small at light load and becomes large at high load.

From these two graphs, it is clear that a lower average can be obtained by using smaller packets at the cost of reduced channel utilization.

In the substation real-time environment, the average load is not high and the data packet size is less than 512 bits. From the delay-load graph, it is shown that the average delay time is less than 1.0 ms. This 1.0 ms delay time is well within the estimated allowable range of message delay time of 33 ms in section 5.2. This indicates that the MPCD network can satisfy the real-time constraints in the substation environment.

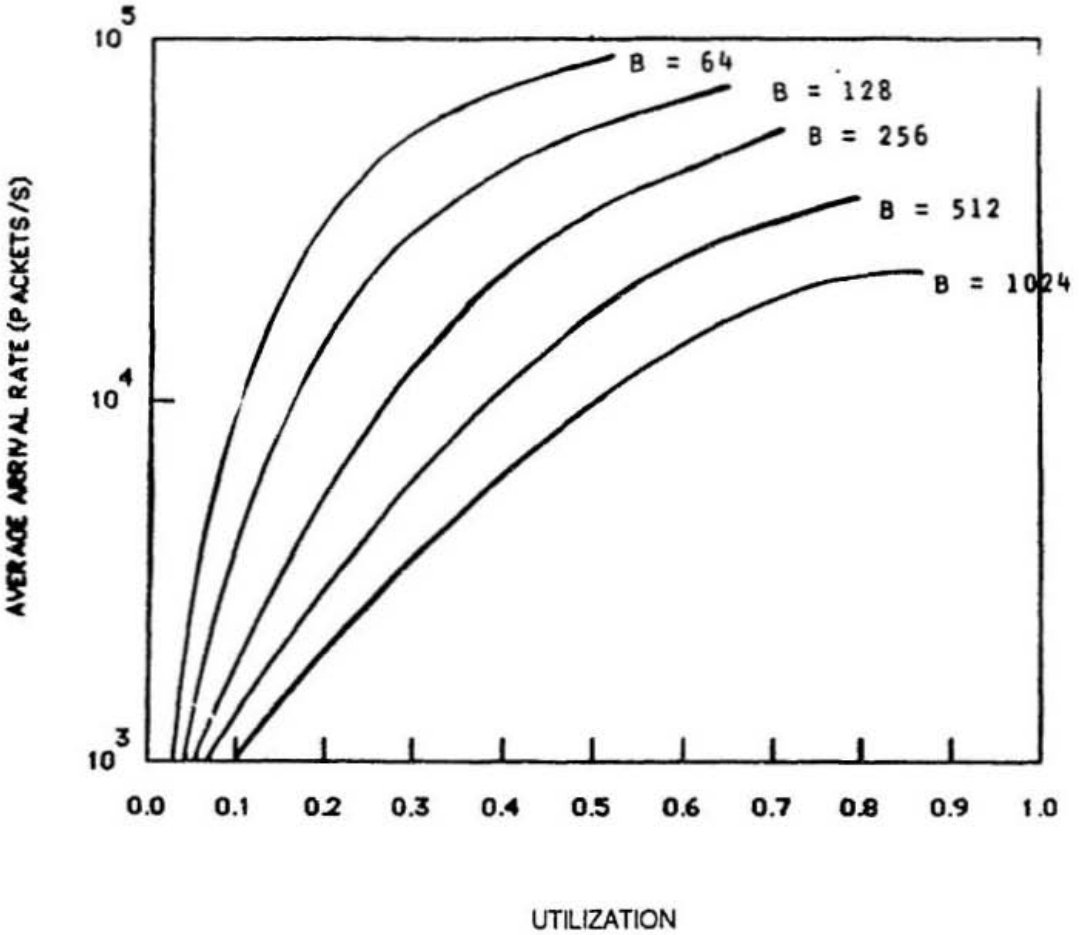

Figure 6.2. Channel Utilization at different load factor (B = packet size).

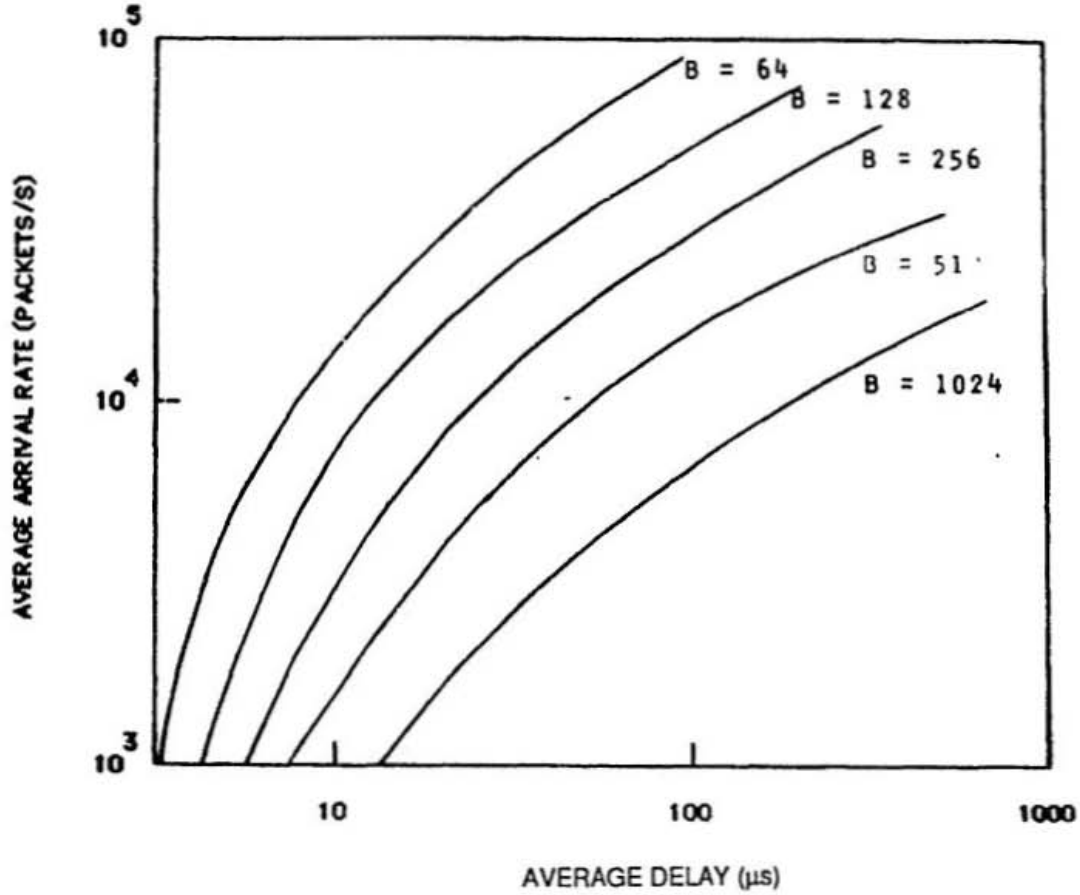

Figure 6.3. Average delay for various loads (B = packet size).

6.3.5 COMPARISON WITH NOOR'S ASCD NETWORK [12]

The average delay vs load relationship for Noor's ASCD protocol is plotted against the MPCD protocol in figure 6.4.

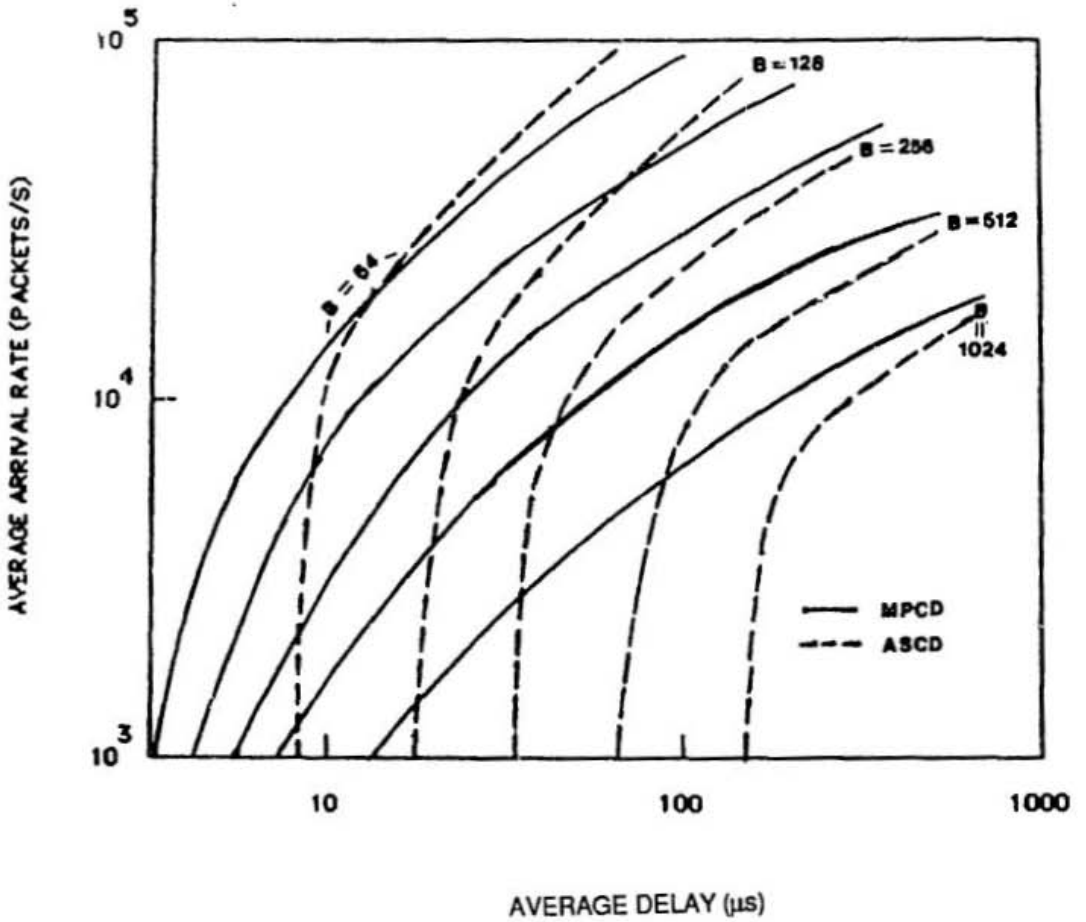

Figure 6.4. Average delay comparison of MPCD and ASCD protocol (B = packet size).

At light traffic load, it is clear that the average delay of the MPCD protocol is considerably shorter than the ASCD protocol. This is due to variable reservation slot time in the MPCD protocol. The ASCD protocol has a fixed reservation slot time, the delay time is therefore at least one fixed slot time plus the message delay time. Whereas the MPCD protocol has a variable reservation slot time, the reservation time is depended on the priority level of the incoming data, the higher the priority, the smaller the reservation time, and therfore the shorter is the delay time.

At high load, the average delay times of the two protocol are comparable in the same range. The reason being that the message delay time is much larger than the reservation slot time at high load resulting from the data collision, therfore the effect of reducing reservation slot time is not significant.

Figure 6.5 illustrates the load-utilization relationship of the two protocols. The graph shows the ASCD has a slightly higher channel utilization than the MPCD protocol under the same load condition. Noor [12] also concluded that the upper limit of the channel utilization of the ASCD protocol is over 95% whereas the upper limit of channel utilization of the MPCD protocol is about 90%.

6.3.6 SUMMARY

A simulation model has been developed to study the performance of the MPCD protocol. A number of observations are made in this analysis:

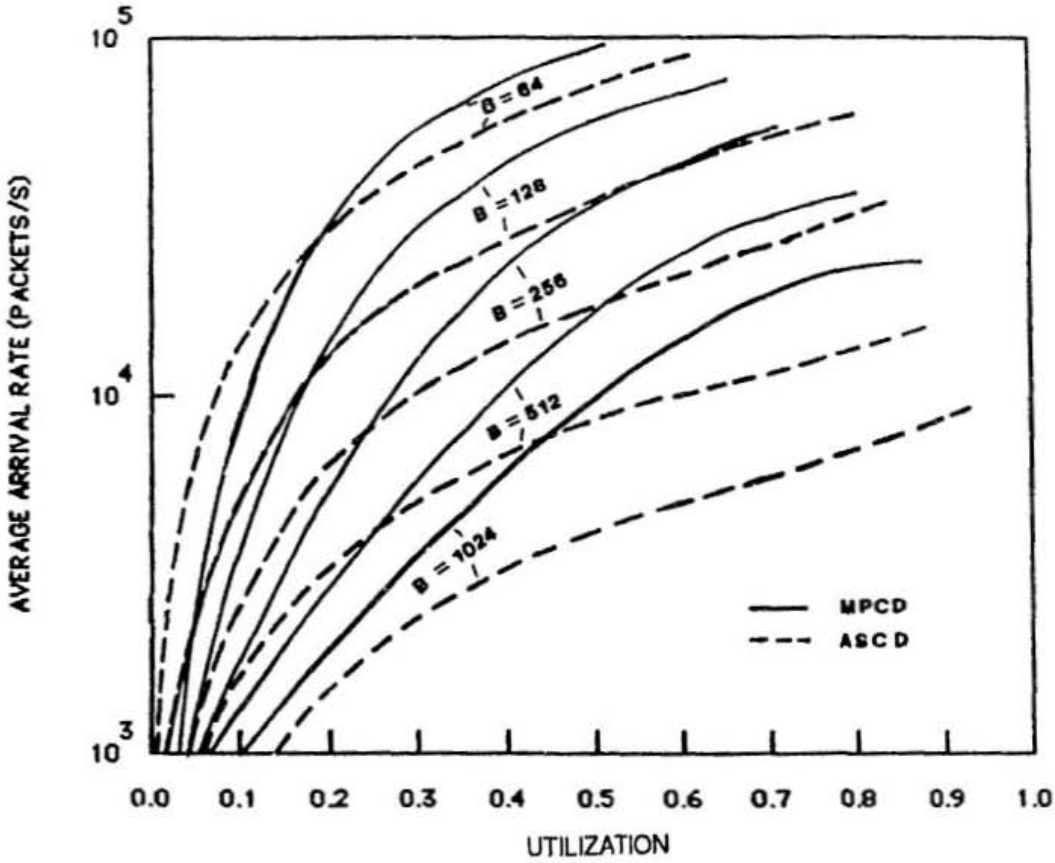

Figure 6.5. Channel utilization comparison of MPCD & ASCD protocol (B = packet size).

- The MPCD network is very stable, i.e. channel utilization is a non-decreasing function of load.
- The MPCD protocol has a better performance than the CSMA type of protocols.
- The performance of the MPCD protocol is sensitive to the packet size.
- The MPCD protocol can satisfy the real-time constraints in the substation environment.
- The MPCD protocol has a better delay-load relationship than the ASCD protocol.
- The ASCD protocol has a slightly higher channel utilization than the MPCD protocol.

CHAPTER 7
CONCLUSIONS

The purpose of this research has been to design a distributed microcomputer network for a real-time controlled environment such as the digital protection of a power substation. The major concern here has been the response time of the data. A network was proposed and simulated under specified conditions. It was found that the network can satisfy the time constraint requirement in the substation environment.

7.1 MAIN CONTRIBUTIONS

A new design concept of a totally distributed local computer network has been presented. The totally distributed approach ensures a completely modular design which simplifies the whole system design work. The distributed scheme also provides a flexible and expandable system configuration.

The integrated protection and control system replaces the mass of switchyard wiring and discrete control-room equipment currently in use. The network shares input data and computes intermediate results, while it controls output facilities to provide more intelligent and sophisticated functions than in the past by the use of dedicated and isolated microcomputers.

The proposed message priority protocol achieves high channel utilization, guarantees access to the network and above all meets the real-time constraints.

7.2 SUGGESTIONS FOR FUTURE RESEARCH

Further development is necessary to implement the network with actual hardware and software to operate in a real life substation environment. Additional work will be required in the development of the network interfaces as well as in the verifications of the reliability and availability of the network.

The integrated digital protection scheme is of interest to replace the existing bulky relaying devices. This area warrants further research work.

The MPCD network is designed for the substation control. However, by following these guidelines, the MPCD protocol can also be implemented in other real-time applications like process control and office automation,etc. Verification of the simulation results using queueing theory is proposed.

REFERENCES

1. Bell, J.R. "Future directions in computing", Computer Design, pp. 95-102, March 1981.

2. Black, U.D. "Data communications, networks and distributed processing", Prentice-Hall, Reston, Virginia, 1983.

3. Deliyanides, J.S. Kezunovic, M. and Schwalenstoker T.H. "An integrated microprocessor based substation protection and control system", IEE conference on Power System Monitoring and Control, pp. 50-55, June 24-26, 1980.

4. Rummer, D.I. and Kezunovic, M. "A Distributed processing Microprocessor based Hierarchically structured network (DMH) with applications to the control and protection of electric power systems", ISMM sixth International symposium and exhibition, MIMI 1978, June 1978.

5. Rummer, D.I. and Kezunovic, M. "Microprocessor systems and architectures for applications to the control and protection of electric power systems", ACM symposium SIGSMALL, Dallas, pp. 80-90, 1979.

6. Takana, K. Kanou, K. Harumoto, Y. Mori, T. Suzuki, K. and Goudo, T. "Application of microprocessors to the control and protection system at substation", IEEE PES Summer Meeting, Paper No. F79 629-7, Vancouver, B.C., 1979.

7. Nilsson, S.L. "ERPI reserach and development of new substation control and protection equipment", IEE conference on developments in Power system protection, London, 1980.

8. Deliyannides, J.S. and Udren, E.A. "Design criteria for an integrated microprocessor-based substation protection and control system", IEEE transactions on PAS, Vol. PAS-101, No. 6, pp. 1664-1673, June 1982.

9. Udren E.A. and Sackin, M., "Relaying features of an integrated microprocessor-based substation controland protection system", IEE conference on Power System Protection, pp. 97-101, June 10-12, 1980.

10. Westinghouse Electric Corporation "WESPAC substsation and control" descriptive bulletin 21-707, April 1984.

11. Westinghouse Electric corporation, "An integrated, microprocessor-based system for relaying and control of substation -- design features and testing program", 12th Annual Western protective relay conference, SPOKENE, Westinghouse automated department, 1985.

12. Noor, A.I. "A microprossor network for the real time processing requirements in a substation", Ph.D. thesis, department of electrical engineering, University of Calgary, Calgary, Alberta, October 1982.

13. Puri, D. Hardy, R.H.S. Davall P.W. Kennedy, W.O. McCuskee, L.B. and Misskey, W.J. "A local area network for an electrical substation", IEEE transactions on PAS, Vol. PAS-103, No. 9, pp. 2399-2404, September 1984.

14. Hope, G.S. and Malik, O.P. "Proposed design philosophy for a microprocessor based substation monitoring and control system", IEE international conference on power system monitoring and control, London, June 24-26, 1980.

15. Damsker, D. "Totally distributed, redundantly structured hardware and software local computer control network", IEEE transactions on PAS, Vol. PAS-102, No.1, pp. 127-133, January 1983.

16. Cory, J. and Moont J.F. "Application of digital computers to protection", IEEE conference on application of computers to power system protection and control meeting, Bournemouth, England, May 1970.

17. Jeyasurya, B. and Rahman, M.A. "A comparative study of transformer protection algorithms", Canadian Electrical Association, Spring meeting, Toronto, Ontario, 1986.

18. Malik, O.P. "Digital protection of a power transformer", IEEE publication 76Ch1075-1 PWR, paper no. A76 191-7, IEEE PES Winter meeting, New York, pp. 1-7, January 1976.

19. Degens, A.J. "Algorithm for a digital transformer differential protection based on least-squares curve fitting", proceedings IEE (London), Vol.128, part C, No.3, pp. 155-161, May 1981.

20. Rahman, M.A. Dash, P.K. and Downton, E.R. "Digital protection of power transformer based on weighted least squares curve fitting", IEEE transactions on PAS, Vol. PAS-101, No.11, pp. 4202-4209, November 1982.

21. Rahman, M.A. and Dash, P.K. "Fast algorithm for digital protection of power transformer", proceedings IEE (London), Vol.129, part C, No.2, pp. 79-85, March 1982.

22. Rahman, M.A. Jeyasurya B. and Gangopadhyay "Digital differential protection of power transformers based on Walsh functions", transactions of CEA Engineering and operating division, Vol.24, part C, paper 85-sp-149, 1985.

23. The Art of Protective Relaying, General Electric, power system management department, Philadephia, Pennsylvania, 1973.

24. Rockefeller, G.D. "Fault protection with a digital computer", IEEE transactions on PAS, Vol. 88, pp. 438-462, April 1969.

25. Malik, O.P. "Schemes for digital protection of transmission lines", Canadian Electrical Association, Spring meeting, Toronto, Ontario, 1986.

26. Fakruddin, D.B. Parthasarathy, K. Jenkins, L. and Hogg, B.W. "Application of Haar functions for transmission line and transformer differnetial protection", Electrical power and energy systems, Vol.6, No.3, pp. 169-180, July 1984.

27. "SCADA - An exploding technology", Electric World, pp. 27-38, October 1976.

28. Flint. D.C. "The data ring man - An introduction to local area networks", Computer Science Series, John Wiley & sons, 1983.

29. Tropper, C. "Local computer network technologies", Academic Press, 1981.

30. ARPANET protocol handbook, Network Information Center, SRI International, Menlo Park, California, January 1978.

31. Crocker, S.D. Heafner, J.F. Metcalfe, R.M. and Postel, J.B. "Function oriented protocols for the ARPA computer network", AFIPS conference proceedings, Vol.40, 1972 SJCC, AFPIS Press, Montvale, N.J., pp. 161-175, 1972.

32. Abramson, N. "The ALOHA system", AFIPS conference proceedings, Vol.37, 1970 FJCC, AFPIS Press, Montvale, N.J., pp. 281-285, 1970.

33. Abramson, N. and Kuo, F.F. "Computer communication networks", Prentice-Hall, Englewood Cliffs, N.J., 1975.

34. Willard, D.G. "MITRIX, a sophisticated digital cable communications system", proceedings of the national telecommunications confernece, November 1973.

35. Farber, D.J. et al "The distributed computing system", proceedings of the 7th annual IEEE computer society international conference, February 1973.

36. Metcalfe, R.M. and Boggs D.R. "ETHERNET: distributed packet switching for local computer networks", communications of the ACM, Vol.19, No.17, pp. 395-404, July 1976.

37. Stallings, W. "Local networks", Computing Surveys, Vol.16, No.1, pp. 3-41, March 1984.

38. Allan, R. "Local networks: fiber optics gains momentum", Electronics Des., June 1983.

39. Tanenbaum, A.S. "Computer networks", Prentice-Hall, second edition, pp. 127-148, 1988.

40. Stallings, W. "A tutorial on the IEEE 802 local network standard", local area and multiple access network, Computer Science Press, pp. 1-30, 1986.

41. White, J.A., Schmidt, J.W. and Bennett, G.K. "Analysis of queueing systems", Acadamic Press, 1975.

42. Kleinrock, L. "Queueing theory, Volume 1 : Theory", New York, John Wiley, 1976.

43. Kleinrock, L. "Queueing theory, Volume 2 : Computer applications", New York, John Wiley, 1976.

44. Tobagi, F.A. and Kleinrock, L. "Packet switching in radio channels : Part II - hidden terminal problem in carrier sense sense multiple access and the busy tone solution", IEEE Transactions on Communications, Vol. COM-23, pp. 1417-1433, December 1975.

45. Lam, S.S. "A carrier sense multiple access protocol for local area networks", Computer network, Vol. 4, pp. 21-32, 1980.

46. Yeh, J.W. "Simulation of local computer networks -- A case study", computer Networks, pp. 401-417, 3 1979.

47. Yeh, J.W. and Donnelley, J.E. "Simulation studies of round robin contention in a prioritized CSMA broadcast network", Proceedings of third conference local computer networking, Minneapolis, October 1978.

48. Yeh, J.W. and Donnelley, J.E. "Interaction between protocol levels in prioritized CSMA broadcast network", Computer network, pp. 9-23, 3 1979.

49. Law, A.M. and Kelton, W.D. "Simulation modelling and analysis", McGraw-Hill book company, 1982.

50 MacDougall, M.H. "Simulating Computer Systems : Techniques and Tools", Computer Systems Series, The MIT Press, 1987.

51. McGovern, T "Data Communications Concepts and Applications", Prentice-Hall, pp.31-36, 1988.

Appendices

A. Listing of the MPCD Simulation Model

B. Listing of the SMPL Simulation Environment

Appendix A

Listing of the MPCD Simulation Model

```
/*****************************************************************
 *                                                               *
 *      Name     : MPCD.C                                        *
 *      Function : Message priority CSMA network simulation model *
 *                                                               *
 *****************************************************************/
#include <queue.h>

#define busy    1
#define idle    0
#define Na      200             /* max. no. of active stations  */
#define Tp      0.0225          /* propagation delays (ms)      */
#define Tif     0.0096          /* interframe delay (ms)        */
#define Tslot   0.0512          /* slot time (ms)               */
#define Tjam    0.0032          /* jam time (ms)                */

struct request  {               /* trans. request descriptor    */
    int pri;                    /* priority class               */
    int attempt;                /* no. retransmission attempts  */
    int bkf;                    /* current backoff count        */
    int lnk;                    /* avail/defer wait list link   */
    real tin;                   /* request initiation time      */
    real txf;                   /* request's transmission time  */
} desc[Na+1];

int N=200, chnl=idle, avl=1, dfr=0, end=0;
/* N=no. of station in the network, chnl=channel status,     */
/* avl=avail. descriptor list end, dfr=defer wait list head, */
/* end=run termination flag                                  */

real a, G, Tf, Ti, tbs=0.0, tis=0.0, tfsum=0.0;
/* a=propagation/transfer ratio, G=offered load=N*Tf/(Ti+Tf),  */
/* Tf= mean frame trans. time, Ti=mean inter-request time/stn, */
/* tbs=channel busy start times, tis=channel idle start times, */
/* tfsum= frame transmission time sum                          */

/*****************************************************************
 *                                                               *
 *      Name     : MAIN                                          *
 *      Function : This is the main program for the MPCD model   *
 *                                                               *
 *****************************************************************/
main()
{
    int event,stn,nb,bus;
    real hw,Td;
    FILE *fp;
    printf("Enter propagation delay/transfer time ratio (a) : ");
    scanf("%lf",&a);
    printf("Enter offered load (G) : ");
    scanf("%lf",&G);
    Tf=Tp/a;
    Ti=Tf*((real)N/G-1.0);
    fp=fopen("SIMUL.LIS","w");
    simul("MPCD simulation model",fp,0);
    bus=facility("BUS",1);
    for (stn=1; stn<=Na; stn++)
        desc[stn].lnk=stn+1;
    desc[Na].lnk=0;
    init_bm(2000,2000);
```

```
    schedule(1,Tp,0);
    while (!end) {      /* run until 10% Td accuracy achieved */
        cause(&event,&stn);
        switch(event) {
            case 1  : TransmitFrame();
                      break;
            case 2  : Defer(stn);
                      break;
            case 3  : StartTransmit(stn);
                      break;
            case 4  : EndTransmit(stn);
                      break;
            case 5  : InitBackoff(stn);
                      break;
            case 6  : Deassert();
                      break;
        }
    }
    civals(&Td,&hw,&nb);
    printf("For a = %.5f, G = %.5f :\n",a,G);
    printf("S = %.3f\n",tfsum/time());
    printf("D = %.3f +/- %.3f\n",Td/Tf,hw/Tf);
    report();
    fclose(fp);
}

/*****************************************************************************
 *                                                                           *
 *      Name     : DLY                                                       *
 *      Function :                                                           *
 *                                                                           *
 *****************************************************************************/
real dly()
{
    return(Tp*(1.0-sqrt(ranf())));
}

/*****************************************************************************
 *                                                                           *
 *      Name     : MAXM                                                      *
 *      Function :                                                           *
 *                                                                           *
 *****************************************************************************/
real maxm(x,y)
real x,y;
{
    return(x>y? x:y);
}

/*****************************************************************************
 *                                                                           *
 *      Name     : TRANSMITFRAME                                             *
 *      Function :                                                           *
 *                                                                           *
 *****************************************************************************/
TransmitFrame ()
{
    int stn;
    struct request *p;
    if (avl) {  /* allocate & build request descriptor */
```

```
        stn=avl;
        p=&desc[stn];
        avl=p->lnk;
        p->attempt=p->bkf=0;
        p->tin=time();
        p->txf=Tf;
        schedule(2,0.0,stn);
        N--;
    }
    if (N)
        schedule(1,expntl(Ti/N),0);
}

/*****************************************************************
 *                                                               *
 *      Name     : DEFER                                         *
 *      Function :                                               *
 *                                                               *
 *****************************************************************/
Defer(stn)
int stn;
{
    real dt=dly();
    struct request *p=&desc[stn];
    if (chnl==busy && time()>(tbs+dt+Tif)) {
        p->lnk=dfr;
        dfr=stn;
    }
    else
        schedule(3,maxm(tbs+dt+Tif-time(),0.0),stn);
}

/*****************************************************************
 *                                                               *
 *      Name     : STARTTRANSMIT                                 *
 *      Function :                                               *
 *                                                               *
 *****************************************************************/
StartTransmit (stn)
int stn;
{
    real tc,t=time(),dt;
    static int ncls;
    struct request *p=&desc[stn];
    if (chnl==idle) {   /* reserve channel & schedule EndTransmit */
        chnl=busy;
        tbs=t;
        ncls=0;
        schedule(4,p->txf,stn);
    }
    else {              /* collision will occur in tc ms. */
        dt=dly();
        tc=maxm(tbs+dt-t,0.0);
        if (++ncls==1) {        /* cancel EndTransmit event */
            schedule(5,Tjam+dt,cancel(4));
            schedule(6,maxm(Tjam+dt,tbs+Tp-t),0);
        }
        schedule(5,Tjam+tc,stn);
    }
}
```

```
/*****************************************************************
 *                                                               *
 *      Name      : ENDTRANSMIT                                  *
 *      Function :                                               *
 *                                                               *
 *****************************************************************/
EndTransmit (stn)
int stn;
{   /* end successful frame transmission */
    struct request *p=&desc[stn];
    tfsum+=p->txf;
    EndRequest(stn);
    schedule(6,0.0,0);
}

/*****************************************************************
 *                                                               *
 *      Name      : InitBackoff                                  *
 *      Function :                                               *
 *                                                               *
 *****************************************************************/
InitBackoff (stn)
int stn;
{
    int k;
    struct request *p=&desc[stn];
    if (++p->attempt>16)                /* abandon request */
        EndRequest(stn);
    else {      /* compute and schedule backoff delay */
        if (p->attempt==1)
            p->bkf=2;
        else if (p->bkf<1024)
            p->bkf*=2;
        if ((k=random(0,p->bkf-1))==0) {
            p->lnk=dfr;
            dfr=stn;
        }
        else
            schedule(2,Tslot*k,stn);
    }
}

/*****************************************************************
 *                                                               *
 *      Name      : DEASSERT                                     *
 *      Function :                                               *
 *                                                               *
 *****************************************************************/
Deassert ()
{
    chnl=idle;
    tis=time();
    while (dfr) {       /* activate requests on defer wait list */
        schedule(3,dly()+Tif,dfr);
        dfr=desc[dfr].lnk;
    }
}

/*****************************************************************
```

```
*                                                                    *
*     Name     : ENDREQUEST                                          *
*     Function :                                                     *
*                                                                    *
**********************************************************************/
EndRequest (stn)
int stn;
{   /* deallocate descriptor, reschedule next arrival */
    struct request *p=&desc[stn];
    end=obs(time()-p->tin);
    p->ink=avl;
    avl=stn;
    N++;
    cancel(1);
    schedule(1,expnti(Ti/N),0);
}
```

Appendix B

Listing of the SMPL Simulation Environment

```
/*****************************************************************
 *                                                               *
 *      Name : QUEUE.C                                           *
 *      Function : This is the queueing simulation subsystem     *
 *                                                               *
 *****************************************************************/
#include <stdio.h>

typedef double real;

#define nl 256                  /* element pool length          */
#define ns 256                  /* namespace length             */
#define pl 58                   /* printer page length          */
#define sl 23                   /* screen page length           */
#define FF 12                   /* form feed                    */

static FILE
    *display=stdout,            /* screen display file          */
    *opf=stdout;                /* current output destination   */

static int
    event,                      /* current simulation event     */
    token,                      /* last token dispatched        */
    blk,                        /* next available block index   */
    avl,                        /* available element list header */
    evl,                        /* event list header            */
    fchn,                       /* facility descriptor chain header */
    avn,                        /* next available namespace pos. */
    tr,                         /* event trace flag             */
    lft=sl;                     /* lines left on current page/scr. */

static real
    clock,                      /* current simulation time      */
    start,                      /* simuation interval start time */
    tl;                         /* last trace message issue time */

static int
    l1[nl],                     /*      facility descriptor     */
    l2[nl],                     /*            queue             */
    l3[nl];                     /*              &               */
static real                     /*         event list           */
    l4[nl],                     /*        element pool          */
    l5[nl];                     /*                              */

static char
    name[ns];                   /* model and facility name space */


/*****************************************************************
 *                                                               *
 *      Name     : SIMUL                                         *
 *      Function : This program initialize the simulation subsystem *
 *                                                               *
 *****************************************************************/
simul(s,fp,j)
int j;
char *s;
FILE *fp;
{
    int i;
```

```
    static int rns=1;
    opf=fp;                         /* assign output file name */
    blk=1; avl=-1; avn=0;           /* element pool & namespace headers */
    evl=fchn=0;                     /* event list & descriptor chain headers */
    clock=start=tl=0.0;             /* sim., interval start, last trace times */
    event=0;tr=j;                   /* current event no. & trace flags */
    for (i=0; i<nl; i++) {
        l1[i]=l2[i]=l3[i]=0;
        l4[i]=l5[i]=0.0;
    }
    i=save_name(s,50);              /* model name->namespace */
    rns=stream(rns);
    rns=++rns>15? 1:rns;            /* set random number stream */
}

/*****************************************************************
 *                                                               *
 *      Name     : RESET                                         *
 *      Function : This routine reset the measurements           *
 *                                                               *
 *****************************************************************/
reset()
{
    resetf();
    start=clock;
}

/*****************************************************************
 *                                                               *
 *      Name     : SAVE_NAME                                     *
 *      Function : This routine saves the model name             *
 *                                                               *
 *****************************************************************/
static save_name(s,m)
char *s;
int m;
{
    int i,n;
    n=strlen(s);
    if (n>m)
        n=m;
    if (avn+n>rns)                  /* namespace exhaused */
        error(2,"O");
    i=avn;
    avn+=n+1;
    strncpy(&name[i],s,n);
    if (n<m)
        name[avn++]='\0';
    return(i);
}

/*****************************************************************
 *                                                               *
 *      Name     : *MNAME                                        *
 *      Function : This functiion get the model name             *
 *                                                               *
 *****************************************************************/
char *mname()
{
    return(name);
```

```
}

/*****************************************************************
 *                                                               *
 *     Name     : *FNAME                                         *
 *     Function : This function get the facility name            *
 *                                                               *
 *****************************************************************/
char *fname(f)
int f;
{
    return(&name[13[f+1]]);
}

/*****************************************************************
 *                                                               *
 *     Name     : GET_BLK                                        *
 *     Function : This routine get a block request               *
 *                                                               *
 *****************************************************************/
static get_blk(n)
int n;
{
    int i;
    if (blk==0)                 /* block request after schedule */
       error(3,"0");
    i=blk;
    blk+=n;
    if (blk>=nl)                /* element pool exhaused */
       error(1,"0");
    return(i);
}

/*****************************************************************
 *                                                               *
 *     Name     : GET_ELM                                        *
 *     Function : This routine get a element from list           *
 *                                                               *
 *****************************************************************/
static get_elm()
{
    int i;
    if (avl<=0) {
       if (avl==0)              /* empty element list */
           error(1,"1");
       /* building the free element list from the block of elements */
       /* remaining after all facilities have been defined */
       for (i=blk; i<(nl-1); i++)
           l1[i]=i+1;
       avl=blk;
       blk=0;
    }
    i=avl;
    avl=l1[i];
    return(i);
}

/*****************************************************************
 *                                                               *
 *     Name     : PUT_ELM                                        *
```

```
 *      Function : This routine return a element                        *
 *                                                                      *
 ************************************************************************/
static put_elm(i)
int i;
{
    l1[i]=avl;
    avl=i;
}

/************************************************************************
 *                                                                      *
 *      Name     : SCHEDULE                                             *
 *      Function : This routine schedules the next event                *
 *                                                                      *
 ************************************************************************/
schedule(ev,te,tkn)
int ev,tkn;
real te;
{
    int i;
    if (te<0.0)         /* negative event time */
        error(4,"0");
    i=get_elm();
    l2[i]=tkn;
    l3[i]=ev;
    l4[i]=0.0;
    l5[i]=clock+te;
    enlist(&evl,i);
    if (tr)
       msg(1,tkn,"",ev,0);
}

/************************************************************************
 *                                                                      *
 *      Name     : CAUSE                                                *
 *      Function : This routine causes a event to execute               *
 *                 from the event list                                  *
 *                                                                      *
 ************************************************************************/
cause (ev,tkn)
int *ev,*tkn;
{
    int i;
    if (evl==0)         /* empty event list */
        error(5,"0");
    i=evl;
    *tkn=token=l2[i];
    *ev=event=l3[i];
    clock=l5[i];
    evl=l1[i];          /* delink element and */
    put_elm(i);         /* return to pool */
    if (tr)
        msg(2,*tkn,"",event,0);
}

/************************************************************************
 *                                                                      *
 *      Name     : TIME                                                 *
 *      Function : This function return the current time                *
```

```
 *                                                                    *
 **********************************************************************/
real time ()
{
    return(clock);
}

/**********************************************************************
 *                                                                    *
 *      Name     : CANCEL                                             *
 *      Function : This routine cancel an event from the event list   *
 *                                                                    *
 **********************************************************************/
cancel(ev)
int ev;
{
    int pred,succ=evl,tkn;
    while ((succ!=0) && (l3[succ]!=ev)) {
        pred=succ;
        succ=l1[pred];
    }
    if (succ==0)
        return(-1);
    tkn=l2[succ];
    if (tr)
        msg(3,tkn,"",l3[succ],0);
    if (succ==evl)              /* unlink event list entry and */
        evl=l1[succ];
    else
        l1[pred]=l1[succ];
    put_elm(succ);              /* deallocate it */
    return(tkn);
}

/**********************************************************************
 *                                                                    *
 *      Name     : SUSPEND                                            *
 *      Function : This routine suspend an event in the event list    *
 *                                                                    *
 **********************************************************************/
static suspend(tkn)
int tkn;
{
    int pred,succ=evl;
    while ((succ!=0) && (l2[succ]!=tkn)) {
        pred=succ;
        succ=l1[pred];
    }
    if (succ==0)                /* no event scheduled for token */
        error(6,"0");
    if (succ==evl)              /* unlink event list entry */
        evl=l1[succ];
    else
        l1[pred]=l1[succ];
    if (tr)
        msg(6,-1,"",l3[succ],0);
    return(succ);
}

/**********************************************************************
```

```
*                                                                    *
*      Name     : ENLIST                                             *
*      Function : This routine inserts the leement in the queue      *
*                 or the event list                                  *
*                 - 'head' points to head of queue/event list        *
*                                                                    *
*********************************************************************/
static enlist(head,elm)
int *head,elm;
{
    int pred,succ;
    real arg,v;
    arg=l5[elm];
    succ=*head;
    while(1) { /* scan for position to insert entry :                */
               /* - event list is ordered in ascending 'arg' values, */
               /*   queues in ascending order                        */
        if (succ==0)            /* end of list */
            break;
        else {
            v=l5[succ];
            if (*head==evl) {   /* event list */
                if (v>arg)
                    break;
            }
            else { /* queue : if entry is for a preempted token  */
                   /* (l4, the remaining event time, >0), insert */
                   /* entry at beginning of its priority class;  */
                   /* otherwise, insert it at the end            */
                if ((v<arg) || ((v==arg) && (l4[elm]>0.0)))
                    break;
            }
        }
        pred=succ;
        succ=l1[pred];
    }
    l1[elm]=succ;
    if (succ!=*head)
        l1[pred]=elm;
    else
        *head=elm;
}

/*********************************************************************
*                                                                    *
*      Name     : FACILITY                                           *
*      Function : This routine defines a facility representing       *
*                 the simulation system's server                     *
*                                                                    *
*********************************************************************/
facility(s,n)
char *s;
int n;
{
    int f,i;
    f=get_blk(n*2);
    l1[f]=n;
    l3[f+1]=save_name(s,(n>1 ? 14:17));
    if (fchn==0)
        fchn=f;
```

```
    else {
        i=fchn;
        while (l2[i+1])
            i=l2[i+1];
        l2[i+1]=f;
    }
    l2[f+1]=0;
    if (tr)
        msg(13,-1,fname(f),f,0);
    return(f);
}

/*****************************************************************
 *                                                               *
 *      Name     : RESETF                                        *
 *      Function : This routine reset the facility ras well as   *
 *                 the queue measurements                        *
 *                                                               *
 *****************************************************************/
static resetf()
{
    int i=fchn,j;
    while (i) {
        l4[i]=l4[i+1]=l5[i+1]=0.0;
        for (j=i+2; j<=(i+l1[i]+1); j++) {
            l3[j]=0;
            l4[j]=0.0;
        }
        i=l2[i+1];              /* advance to next facility */
    }
    start=clock;
}

/*****************************************************************
 *                                                               *
 *      Name     : REQUEST                                       *
 *      Function : This routine requests for the facility        *
 *                                                               *
 *****************************************************************/
request(f,tkn,pri)
int f,tkn,pri;
{
    int i,r;
    if (l2[f]<l1[f]) {              /* facility nonbusy : reserves */
        for (i=f+2; l1[i];i=0; i++);  /* 1st found nonbusy server */
        l1[i]=tkn;
        l2[i]=pri;
        l5[i]=clock;
        l2[f]++;
        r=0;
    }
    else {                          /* facility busy : enqueue token */
        enqueue(f,tkn,pri,event,0.0);   /* marked w/event, priority */
        r=1;
    }
    if (tr)
        msg(7,tkn,fname(f),r,l3[f]);
    return(r);
}
```

```
/*****************************************************************
 *                                                               *
 *      Name     : ENQUEUE                                       *
 *      Function : This routine enqueue a token                  *
 *                                                               *
 *****************************************************************/
static enqueue(f,j,pri,ev,te)
int f,j,pri,ev;
real te;
{
    int i;
    l5[f+1]+=l3[f]*(clock-l5[f]);
    l3[f]++;
    l5[f]=clock;
    i=get_elm();
    l2[i]=j;
    l3[i]=ev;
    l4[i]=te;
    l5[i]=(real)pri;
    enlist(&l1[f+1],i);
}

/*****************************************************************
 *                                                               *
 *      Name     : PREEMPT                                       *
 *      Function : This routine preempt a facility               *
 *                                                               *
 *****************************************************************/
preempt(f,tkn,pri)
int f,tkn,pri;
{
    int ev,i,j,k,r;
    real te;
    if (l2[f]<l1[f]) {              /* facility nonbusy : locate */
        for (k=f+2; l1[k]!=0; k++);  /* 1st found nonbusy server  */
        r=0;
        if (tr)
            msg(8,tkn,fname(f),0,0);
    }
    else {                          /* facility busy : find server */
        k=f+2;                      /* with lowest priority user   */
        j=l1[f]+f+1;                /* indices of server element 1 & n */
        for (i=f+2; i<=j; i++)
            if (l2[i]<l2[k])
                k=i;
        if (pri<=l2[k]) {           /* requesting token's priority is */
            r=1;                    /* not higher than that of anu user */
            enqueue(f,tkn,pri,event,0.0); /* enqueue requester & r=1 */
            if (tr)
                msg(7,tkn,fname(f),1,l3[f]);
        }
        else {              /* preempt user of server k. suspend event,  */
            if (tr)         /* save event no. & remaining event time, &  */
                msg(8,tkn,fname(f),2,0); /* enqueue preempted token. */
            j=l1[k];        /* if remaining event time is 0 (preemption  */
            i=suspend(j);           /* occuried at the instant release   */
            ev=l3[i];       /* was to occur, set 'te' > 0 for proper     */
            te=l5[i]-clock;         /* enqueueing (see ENLIST)). Update  */
            if (te==0.0)            /* facility & server statistics for  */
                te=1.0e-99;         /* for the preempted token, and set  */
```

```
            put_elm(i); /* r=0 to reserve the facility for the       */
            enqueue(f,j,l2[k],ev,te);   /* preempting token.          */
            if (tr) {
                msg(10,-1,"",j,l3[f]);
                msg(12,-1,fname(f),tkn,0);
            }
            l3[k]++;
            l4[k]+=clock-l5[k];
            l2[f]--;
            l4[f+1]++;
            r=0;
        }
    }
    if (r==0) {                 /* reserve server k of facility */
        l1[k]=tkn;
        l2[k]=pri;
        l5[k]=clock;
        l2[f]++;
    }
    return(r);
}

/*************************************************************************
 *                                                                       *
 *      Name     : RELEASE                                               *
 *      Function : This routine release a facility                       *
 *                                                                       *
 *************************************************************************/
release(f,tkn)
int f,tkn;
{
    int i,j=0,k,m;
    real te;
    /* locate server (j) reserved by releasing token */
    k=f+1+l1[f];        /* index of last server element */
    for (i=f+2; i<=k; i++)
        if (l1[i]==tkn) {
            j=i;
            break;
        }
    if (j==0)           /* no server reserved */
        error(7,"0");
    l1[j]=0;
    l3[j]++;
    l4[j]+=clock-l5[j];
    l2[f]--;
    if (tr)
        msg(9,tkn,fname(f),0,0);
    if (l3[f]>0) {      /* queue not empty: dequeue request ('k' =   */
        k=l1[f+1];      /* index of element) & update queue measure */
        l1[f+1]=l1[k];
        te=l4[k];
        l5[f+1]+=l3[f]*(clock-l5[f]);
        l3[f]--;
        l4[f]++;
        l5[f]=clock;
        if (tr)
            msg(11,-1,"",l2[k],l3[f]);
        if (te==0.0) {  /* block request : place request at head of */
            l5[k]=clock;  /* event list (so its facility request */
```

```
            l1[k]=evl;      /* can be re-initiated before any other */
            evl=k;          /* requests scheduled for this time. */
            m=4;
        }
        else {  /* return after preemption : reserve facility for */
            l1[j]=l2[k];        /* dequeued request & reschedule */
            l2[j]=(int)l5[k];   /* remaining event time. */
            l5[j]=clock;
            l2[f]++;
            if (tr)
                msg(12,-1,fname(f),l2[k],0);
            l5[k]=clock+te;
            enlist(&evl,k);
            m=5;
        }
        if (tr)
            msg(m,-1,"",l3[k],0);
    }
}

/*********************************************************************
 *                                                                   *
 *      Name     : STATUS                                            *
 *      Function : This routine get the status of a facility         *
 *                                                                   *
 *********************************************************************/
status(f)
int f;
{
    return(l1[f]==l2[f]? 1:0);
}

/*********************************************************************
 *                                                                   *
 *      Name     : INQ                                               *
 *      Function : This routine get the current queue length         *
 *                                                                   *
 *********************************************************************/
inq (f)
int f;
{
    return(l3[f]);
}

/*********************************************************************
 *                                                                   *
 *      Name     : U                                                 *
 *      Function : This function computes the facility utilization   *
 *                                                                   *
 *********************************************************************/
real U(f)
int f;
{
    int i;
    real b=0.0,t=clock-start;
    if (t>0.0) {
        for (i=f+2; i<=f+l1[f]+1; i++)
            b+=l4[i];
        b/=t;
    }
```

```
    return(b);
}

/*****************************************************************
 *                                                               *
 *      Name     : B                                             *
 *      Function : This function computes the mean busy period   *
 *                                                               *
 *****************************************************************/
real B(f)
int f;
{
    int i,n=0;
    real b=0.0;
    for (i=f+2; i<=f+l1[f]+1; i++) {
        b+=l4[i];
        n+=l3[i];
    }
    return((n>0)? b/n:b);
}

/*****************************************************************
 *                                                               *
 *      Name     : LQ                                            *
 *      Function : This function computes the average queue length *
 *                                                               *
 *****************************************************************/
real Lq(f)
int f;
{
    real t=clock-start;
    return((t>0.0)? (l5[f+1]/t):0.0);
}

/*****************************************************************
 *                                                               *
 *      Name     : TRACE                                         *
 *      Function : This routine turns trace on/off               *
 *                                                               *
 *****************************************************************/
trace(n)
int n;
{
    switch(n) {
        case 0  : tr=0;
                  break;
        case 1  :
        case 2  :
        case 3  : tr=n;
                  tl=-1.0;
                  newpage();
                  break;
        case 4  : end_line();
                  break;
        default : break;
    }
}

/*****************************************************************
 *                                                               *
```

```
 *      Name     : MSG                                                  *
 *      Function : This routine generates trace messages                *
 *                                                                      *
 ************************************************************************/
static msg(n,i,s,q1,q2)
int n,i,q1,q2;
char *s;
{
    static char *m[14] = {"","SCHEDULE","CAUSE","CANCEL",
                          "   RESCHEDULE","   RESUME","   SUSPEND",
                          "REQUEST","PREEMPT","RELEASE","   QUEUE",
                          "   DEQUEUE","   RESERVE","FACILITY"};
    if (clock>tl) {     /* print time stamp (if time has advanced) */
        tl=clock;
        fprintf(opf,"  time %-12.3f  ",clock);
    }
    else
        fprintf(opf,"%21s",m[0]);
    if (i>=0)           /* print token number if specified */
        fprintf(opf,"--  token %-4d  -- ",i);
    else
        fprintf(opf,"--               -- ");
    fprintf(opf,"%s %s",m[n],s);        /* print basic message */
    switch(n) {                 /* append qualifier */
        case 1  :
        case 2  :
        case 3  :
        case 4  :
        case 5  :
        case 6  : fprintf(opf," EVENT %d",q1);
                  break;
        case 7  :
        case 8  : switch(q1) {
                        case 0  : fprintf(opf,":  RESERVED");
                                  break;
                        case 1  : fprintf(opf,":  QUEUED  (inq = %d)",
                                          q2);
                                  break;
                        case 2  : fprintf(opf,":  INTERRUPT");
                                  break;
                        default : break;
                  }
                  break;
        case 9  : break;
        case 10 :
        case 11 : fprintf(opf," token %d (inq = %d)",q1,q2);
                  break;
        case 12 : fprintf(opf," for token %d",q1);
                  break;
        case 13 : fprintf(opf,":  f = %d",q1);
                  break;
        default : break;
    }
    fprintf(opf,"\n");
    end_line();
}

/************************************************************************
 *                                                                      *
 *      Name     : END_LINE                                             *
```

```
 *      Function :                                                      *
 *                                                                      *
 ************************************************************************/
end_line()
{
    if ((--lft)==00) {          /* end of page/screen */
        switch(tr) {
            case 1  : if (opf==display) /* Trace 1 : advance page if */
                          lft=sl;       /* printer output., screen   */
                      else              /* output is free-running.   */
                          endpage();
                      break;
            case 2  : endpage();
                      break;
            case 3  : lft=sl;
                      break;
        }
    }
    if (tr==3)
        pause();
}

/************************************************************************
 *                                                                      *
 *      Name     : PAUSE                                               *
 *      Function : This routine pause execution via 'mtr' call         *
 *                                                                      *
 ************************************************************************/
pause()
{
    getchar();
}

/************************************************************************
 *                                                                      *
 *      Name     : ERROR                                               *
 *      Function : This routine displc, error message & exit           *
 *                                                                      *
 ************************************************************************/
error (n,s)
int n;
char *s;
{
    FILE *dest;
    static char
        *m[8] = {"Simulation Error at Time ",
                 "Empty Element Pool",
                 "Empty Name Space",
                 "Facility Defined After Queue/Schedule",
                 "Negative Event Time",
                 "Empty Event List",
                 "Preempted Token Not in Event List",
                 "Release of idle/Unowned Facilitiy");
    dest=opf;
    while(1) {  /* send message to both printer and screen */
        fprintf(dest,"\n**** %s%.3f\n",m[0],clock);
        if (n)
            fprintf(dest,"     %s\n",m[n]);
        if (*s!='0')
            fprintf(dest,"     %s\n",s);
```

```
        if (dest==display)
            break;
        else
            dest=display;
    }
    if (opfl=display)
        report();
    exit(0);
}

/*****************************************************************
 *                                                               *
 *      Name      : REPORT                                       *
 *      Function : This routine generate the simulation report   *
 *                                                               *
 *****************************************************************/
report()
{
    newpage();
    reportf();
    endpage();
}

/*****************************************************************
 *                                                               *
 *      Name      : REPORTf                                      *
 *      Function : This routine generate the facility report     *
 *                                                               *
 *****************************************************************/
reportf()
{
    int f;
    if ((f=fchn)==0)
        fprintf(opf,"\nno facilities defined :  report abandoned\n");
    else {      /* f = 0 at end of facility chain */
        while(f) {
            f=rept_page(f);
            if (f>0)
                endpage();
        }
    }
}

/*****************************************************************
 *                                                               *
 *      Name      : REPT_PAGE                                    *
 *      Function : This routine generate the report page         *
 *                                                               *
 *****************************************************************/
static rept_page(fnxt)
int fnxt;
{
    int f,i,n;
    char fn[19];
    static char *s[7] = {
        "MPCD simulation report", " MODEL: ", "TIME: ", "INTERVAL: ",
        "MEAN BUSY     MEAN QUEUE        OPERATION COUNTS",
        " FACILITY          UTIL.    ",
        " FACILITY      LENGTH     RELEASE   PREEMPT   QUEUE");
    fprintf(opf,"\n%51s\n\n\n\n",s[0]);
```

```
    fprintf(opf,"%-s%-54s%-s%11.3f\n",s[1],mname(),s[2],clock);
    fprintf(opf,"%68s%11.3f\n\n",s[3],clock-start);
    fprintf(opf,"%75s\n",s[4]);
    fprintf(opf,"%s%s\n",s[5],s[6]);
    f=fnxt;
    lft-=8;
    while (f && lft--) {
        n=0;
        for (i=f+2; i<=f+l1[f]+1; i++)
            n+=l3[i];
        if (l1[f]==1)
            sprintf(fn,"%s",fname(f));
        else
            sprintf(fn,"%s[%d]",fname(f),l1[f]);
        fprintf(opf," %-17s%6.4f %10.3f %13.3f %11d %9d %7d\n",
                fn,U(f),B(f),Lq(f),n,(int)l4[f+1],(int)l4[f]);
        f=l2[f+1];
    }
    return(f);
}

/*****************************************************************
 *                                                               *
 *      Name     : LNS                                           *
 *      Function : This routine counts number of lines in a page  *
 *                                                               *
 *****************************************************************/
lns (i)
int i;
{
    lft-=i;
    if (lft<=0)
        endpage();
    return(lft);
}

/*****************************************************************
 *                                                               *
 *      Name     : ENDPAGE                                       *
 *      Function : This routine print the end of page character   *
 *                                                               *
 *****************************************************************/
endpage()
{
   int c;
    if (opf==display) {         /* screen output :              */
        while (lft>0) {         /* push to top of screen & pause */
            putc('\n',opf);
            lft--;
        }
        printf("\n[ENTER) to continue:");
        getchar();
        printf("\n\n");
    }
    else if (lft<pl)
        putc(FF,opf);
    newpage();
}

/*****************************************************************
```

```
/*                                                                    *
 *      Name     : NEWPAGE                                            *
 *      Function : This routine generates a new report page           *
 *                                                                    *
 **********************************************************************/
newpage()
{ /* set line count to top of page/screen after page change/screen */
  /* clear by SIMUL, another SIMUL module, or simulaton program */
    lft=(opf==display)? sl:pl;
}

/**********************************************************************
 *                                                                    *
 *      Name     : SENDTO                                             *
 *      Function : This routine redirect the output to specfied       *
 *                 destination                                        *
 *                                                                    *
 **********************************************************************/
FILE *sendto(dest)
FILE *dest;
{
    if (dest)
        opf=dest;
    return(opf);
}
```

```
/*****************************************************************
 *                                                               *
 *      Name     : RAND.C                                        *
 *      Function : Random variable generation                    *
 *                                                               *
 *****************************************************************/
#include <math.h>

typedef double real;

#define A 16807L                /* multiplier (7**5) for RANF */
#define M 2147483647L           /* modulus (2**31-1) for RANF */

static long In[16] = {0L,       /* seeds for streams 1 thru 15 */
        1973272912L,747177549L,20464843L,640830765L,1098742207L,
        78126602L,84743774L,831312807L,124667236L,1172177002L,
        1124933064L,1223960546L,1878892440L,1449793615L,553303732L};

static int strm=1;              /* index of current stream */

/*****************************************************************
 *                                                               *
 *      This implementation is for 8086/8 & 80286/386 CPUs using *
 *      complier with 16 bits short & 32 bits long integers      *
 *                                                               *
 *****************************************************************/
real ranf()
{
    short *p,*q,k;
    long Hi,Lo;
    /* generate product using double precision simulation (comments */
    /* refer to In's lower 16 bit as "L", its upper 16 bit as "H")  */
    p=(short *)&In[strm];
    Hi=*(p+1)*A;                /* 16807*H->Hi */
    *(p+1)=0;
    Lo=In[strm]*A;              /* 16807*L->Lo */
    p=(short *)&Lo;             /* add high order bits of Lo to Hi */
    Hi+=*(p+1);
    q=(short *)&Hi;             /* low order bits of Hi -> Lo */
    *(p+1)=*q&0X7FFF;           /* clear sign bit */
    k=*(q+1)<<1;                /* Hi bits 31-45->K */
    if (*q&0X8000)
        k++;
    /* form Z + K [-M] (Z=Lo) : presubtract M to avoid overflow */
    Lo-=M;
    Lo+=k;
    if (Lo<0)
        Lo+=M;
    In[strm]=Lo;
    return((real)Lo*4.656612875E-10);          /* Lo x 1/2(2**31-1) */
}

/*****************************************************************
 *                                                               *
 *      Name     : STREAM                                        *
 *      Function : This routine selects generator stream         *
 *                                                               *
 *****************************************************************/
stream(n)
int n;
```

```
    Hi=*(p+1)*A;                /* 16807*H->Hi */
    *(p+1)=0;
    Lo=In[strm]*A;              /* 16807*L->Lo */
    p=(short *)&Lo;             /* add high order bits of Lo to Hi */
    Hi+=*(p+1);
    q=(short *)&Hi;             /* low order bits of Hi -> Lo */
    *(p+1)=*q&0X7FFF;           /* clear sign bit */
    k=*(q+1)<<1;                /* Hi bits 31-45->K */
    if (*q&0X8000)
        k++;
    /* form Z + K [-M] (Z=Lo) : presubtract M to avoid overflow */
    Lo-=M;
    Lo+=k;
    if (Lo<0)
        Lo+=M;
    In[strm]=Lo;
    return((real)Lo*4.656612875E-10);          /* Lo x 1/2(2**31-1) */
}

/*****************************************************************
 *                                                               *
 *      Name     : STREAM                                        *
 *      Function : This routine selects generator stream         *
 *                                                               *
 *****************************************************************/
stream(n)
int n;
{       /* set stream for 1<=n<=15, return stream for n=0 */
    if ((n<0) || (n>15))
        error(0,"stream argument error");
    if (n)
        strm=n;
    return(strm);
}

/*****************************************************************
 *                                                               *
 *      Name     : SEED                                          *
 *      Function : This function setup/get the seed for the      *
 *                 random generator                              *
 *                                                               *
 ********************* *************************************** */
long seed (lk,n)
long lk;
int n;
{   /* set seeed of stream n for lk>0, return current seed  for lk */
    if ((n<1)||(n>15))
        error(0,"seed argument error");
    if (lk>0L)
        In[n]=lk;
    return(In[n]);
}

/*****************************************************************
 *                                                               *
 *      Name     : UNIFORM                                       *
 *      Function : This function generate a psuedo-random variate *
 *                 from a uniform distribution [a,b]             *
 *                                                               *
 *****************************************************************/
```

```
real uniform(a,b)
real a,b;
{
    if (a>b)
        error(0,"uniform argument error : a > b");
    return(a+(b-a)*ranf());
}

/*****************************************************************
 *                                                               *
 *      Name     : RANDOM                                        *
 *      Function : This routine generate a random integer equip- *
 *                 probably selected from the integer set i,i+1,..n.*
 *                                                               *
 *****************************************************************/
random(i,n)
int i,n;
{
    if (i>n)
        error(0,"Random Argument Error: i > n");
    n-=i;
    n=(n+1.0)*ranf();
    return(i+n);
}

/*****************************************************************
 *                                                               *
 *      Name     : EXPNTL                                        *
 *      Function : This routine generate a psuedo-random variate *
 *                 from a negative exponential distribution with *
 *                 mean x.                                       *
 *                                                               *
 *****************************************************************/
real expntl(x)
real x;
{
    return(-x*log(ranf()));
}

/*****************************************************************
 *                                                               *
 *      Name     : ERLANG                                        *
 *      Function : This routine generate a psuedo-random variate *
 *                 from an erlang distribution with mean x and   *
 *                 standard deviation s.                         *
 *                                                               *
 *****************************************************************/
real erlang(x,s)
real x,s;
{
    int i,k;
    real z;
    if (s>x)
        error(0,"erlang argument error: s > x");
    z=x/s;
    k=(int)z*z;
    z=1.0;
    for (i=0; i<k; i++)
        z*=ranf();
    return(-(x/k)*log(z));
```

```
)

/*****************************************************************
 *                                                               *
 *      Name     : HYPERX                                        *
 *      Function : This routine generate a psuedo-random variate *
 *                 from Morse's two-stage hyperexponential       *
 *                 distribution with mean x and standard deviation *
 *                 s where s>x.                                  *
 *                                                               *
 *****************************************************************/
real hyperx(x,s)
real x,s;
{
    real cv,z,p;
    if (s<=x)
        error(0,"hyperx argument error: s not > x");
    cv=s/x;
    z=cv*cv;
    p=0.5*(1.0-sqrt((z-1.0)/(z+1.0)));
    z=(ranf()>p)? (x/(1.0-p)):(x/p);
    return(-0.5*z*log(ranf()));
}

/*****************************************************************
 *                                                               *
 *      Name     : NORMAL                                        *
 *      Function : This routine generate a psuedo-random variate *
 *                 from a normal distribution with mean x and    *
 *                 standard deviation s                          *
 *                                                               *
 *****************************************************************/
real normal(x,s)
real x,s;
{
    real v1,v2,w,z1;
    static real z2=0.0;
    if (z2!=0.0) {              /* use value from previous call */
        z1=z2;
        z2=0.0;
    }
    else {
        do {
            v1=2.0*ranf()-1.0;
            v2=2.0*ranf()-1.0;
            w=v1*v1+v2*v2;
        } while (w>=1.0);
        w=sqrt((-2.0*log(w))/w);
        z1=v1*w;
        z2=v2*w;
    }
    return(x+z1*s);
}
```