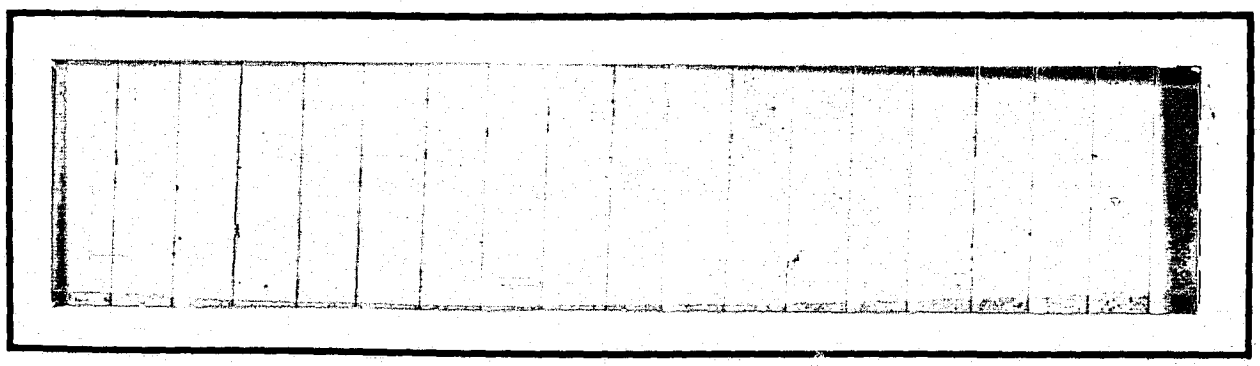

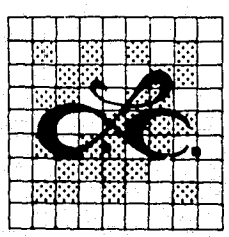

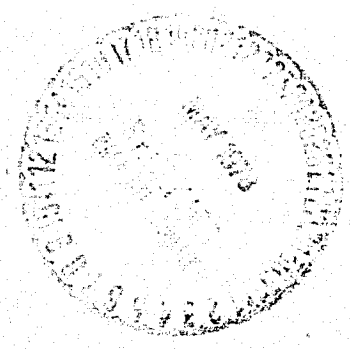

LinCom Corporation
P.O. Box 2793D, Pasadena, Calif. 91105

LinCom

FINAL REPORT

KU-BAND SIGNAL DESIGN STUDY

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
JOHNSON SPACE CENTER
Houston, Texas 77058

Technical Monitor: W. Teasdale

Prepared Under Contract No. NAS 9-14846

Prepared by:

Izhak Rubin
W. C. Lindsey

LINCOM CORPORATION
P.O.Box 2793D
Pasadena, CA 91105

April 15, 1978

TR-7804-0476

LinCom

TABLE OF CONTENTS

	Page
THE ORGANIZATION OF THE REPORT AND SUMMARY	1
I. EVALUATION OF DATA PROCESSING SYSTEM QUEUEING AND SYNCHRONIZATION	5
I.1 The Structure of the Data Processing Network	5
I.1.1 System and Network Configuration and Operation	5
I.1.2 Characteristics of the Computer System	11
I.2 Traffic, Task and Subsystem Models and Parameters	16
I.2.1 The Network Components	16
I.2.2 The Computer Subsystem	16
I.2.3 Terminal, Task and User Traffic	19
I.2.4 Task and Application Process Parameters	23
I.2.5 The Communication Subnetwork	28
I.3 Performance Measures	31
I.3.1 Computer Oriented Performance	31
I.3.2 User Oriented Performance Measures	33
I.3.3 System and Network Related Performance Indices	36
I.4 Time-Sharing Queueing Models	39
I.4.1 Time-Shared Single Processor Systems	39
I.4.2 Traffic and Performance Parameters	40
I.4.3 Batch Processing: First-tone First-Served	42
I.4.4 Round-Robin Processing	45
I.4.5 Round-Robin with Priorities	47
I.4.6 A Round-Robin Scheme with Time-Varying Priorities	49
I.4.7 Foreground-Background Processing Schemes	51

TABLE OF CONTENTS (Cont'd)

		Page
I.4.8	Multilevel Processor Sharing Schemes	53
I.4.9	Comparing the Performance of the Time-Sharing Schemes	54
I.5	Priority Queueing Model	56
I.5.1	On Service Disciplines	56
I.5.2	Scheduling Algorithms for Time-Shared Processing Systems	56
I.5.3	Service Disciplines for Messages in Different Priority Classes	59
I.5.4	Analysis of a Priority Queueing System	60
I.5.5	The Earliest Due Date Scheduling Scheme	65
I.6	The Computer System: Queueing Models and Performance Analysis	74
I.6.1	Operating Systems	74
I.6.2	Memory Management	76
I.6.3	On Computer Scheduling Procedures	80
I.6.4	A Markovian Queueing Model: Finite Buffer Facility	82
I.6.5	A Finite Task Source Queueing Model	86
I.6.6	A Multi-Processor Queueing Model	89
I.6.7	Queueing Models Involving Input/Output and CPU Interactions	92
I.6.8	An Analytical Model for the Computer System Performance Prediction	96
I.7	Queueing Modeling and Analysis Procedures for the Space Shuttle Orbiter Avionics System	103
I.7.1	The Queueing Model	103
I.7.2	A Time Frame Model for the Computer System	104

TABLE OF CONTENTS (Cont'd)

	Page
I.7.3 Queueing Analysis for Cyclic Tasks: Model I	107
I.7.4 Queueing Analysis for Cyclic Tasks: Model II	112
I.7.5 Queueing Analysis for Cyclic Tasks: Model III	115
I.7.6 Queueing Analysis for Acyclic Tasks: Priority Model I	117
I.7.7 Queueing Analysis for Acyclic Tasks: Models II	121
I.7.8 Joint Queueing Analysis	122
I.7.9 Queueing Analysis for User Terminals: Output Traffic	124
I.7.10 Queueing Analysis for User Terminals: Input Traffic	127
I.8 Synchronization Methods for the Data Processing System	130
I.8.1 Synchronization Considerations for the Data Processing System	130
I.8.2 A Queueing Model	135
I.8.3 Clock Synchronization Procedures	136
II. SYSTEM RELIABILITY MEASURES AND COMMUNICATION PATH FAILURE ANALYSIS	147
II.1 Reliability Features of the Data Processing Network	147
II.2 Failure Parameters and Reliability Performance Measures for the Computer Complex	149
II.3 Failure Analysis for the Computer System: The Simplex Mode	156
II.3.1 Single GPC Failure Analysis	156
II.3.2 Failure Analysis for the Simplex Computer System	158

TABLE OF CONTENTS (Cont'd)

	Page
II.3.3 Restoration Analysis for the Simplex Computer System	159
II.4 Failure Analysis for the Computer System: The Redundant Mode	162
II.5 Failure Analysis for an Application Subsystem	174
II.6 Failure Analysis for the Data Processing Network	179
II.6.1 Reliability Performance Measures for the Data Processing Network	179
II.6.2 Failure Analysis for the Data Processing Network: The Redundant Mode	183
II.6.3 Network Invulnerability: Alternate Routing and Congestion Effects	185
II.6.4 Failure Analysis for the Data Processing Network: The Simplex Mode	189
REFERENCES	196

THE ORGANIZATION OF THE REPORT AND SUMMARY

This study provides analytical tools, methods and techniques for assessing the design and performance of the Space Shuttle Orbiter data processing system (DPS). The computer data processing network is evaluated in the following three key areas:

Queueing Behavior;

Synchronization;

Network Reliability.

The report is divided into two main parts. Part I consists of detailed modeling and analyses of queueing and synchronization aspects of the DPS. Part II involves the evaluation of the overall network reliability in the presence of various failure modes. The detailed models, techniques, performance measures and results presented here fully satisfy all the study objectives outlined in the associated technical proposal.

The structure of the data processing network is presented in Section I.1. System operation principles and the network configuration are described. The characteristics of the computer systems are indicated.

Traffic, task and subsystem models and parameters are derived and described in Section I.2. Process parameters and models are presented for the following network elements: the computer subsystem; the terminal, task and user traffic; task and application process parameters; and the communication subnetwork.

The system performance measures are derived, presented and discussed in Section I.3. We differentiate between computer

oriented performance measures, user oriented performance measures and system and network related performance indices.

General important queueing models are described, analyzed and compared in Sections I.4-I.5. Computer system queueing models are presented in Section I.6. Queueing modeling and analysis methods for the orbiter DPS are described in Section I.7.

Time-sharing queueing models are described and analyzed in Section I.4. Included are: time-shared single processor systems; batch processing systems; round-robin processing; round-robin with priorities; a round-robin scheme with time-varying priorities; foreground-background processing shcemes; and multilevel processor sharing scheme. The performance characteristics of the various time-shared schemes are then compared.

Priority queueing models are described and analyzed in Section I.5. While time-sharing schemes increase the operational efficiency of the orbiter computer complex, priority service procedures allow the incorporation of task priorities in providing the proper grade-of-service for critical tasks.

In Section I.6, we present queueing models and demonstrate the performance analysis for the computer system. Operating systems and memory management techniques are discussed. Computer scheduling proceduresare outlined. The following analytical queueing models are then presented, for studying the queueing behavior of the computer system: a Markovian queueing model with finite buffer facility; a finite task source queueing model; a multi-processor queueing model; and queueing models involving input/output (I/O) and CPU interactions.

Queueing modeling and analysis procedures for the Space Shuttle orbiter avionics system are presented in Section I.7. The underlying queueing model is described. A time frame model for the computer system is then chosen. Tasks are divided as being cyclic or acyclic. Proper computer task service times are subsequently allocated. Queueing models are then chosen and analyzed for cyclic and acyclic tasks. Subsequently, the results are integrated to yield a joint queueing model. The latter is analyzed, and the system performance functions are derived, studied and discussed. We then choose proper queueing models for describing message delay and buffer characteristics at the user terminals, considering both input and output traffic.

The synchronization problem is discussed in Section I.8. Synchronization considerations for the data processing system are outlined. A queueing model is presented to relate time offset parameters with message delay and buffer queue-size functions. Clock synchronization procedures are then presented, discussed, compared and analyzed.

In Part II of the report, system reliability measures are defined and studied. System and network invulnerability measures are computed. A communication path and network failure analysis techniques are presented. The reliability features of the data processing network are outlined in Section II.1. In Section II.2 we define failure parameters and reliability performance measures for the computer complex. The failure analysis for the computer system, when operating in the simplex mode, is carried out in SectionII.3. The corresponding failure analysis for the redundant computer system is presented in Section II.4. The invulnerability characteristics and failure

properties of an application subsystem are derived in SectionII.5. These results are integrated and combined in Section II.6, resulting with the failure analysis of the data processing network.

The techniques, methods and results presented in this study are of prime importance as tools in assessing the performance of the orbiter DPS. Furthermore, the models developed and presented here are of general fundamental nature, involving the key aspects of system reliability, queueing (delay-throughput, grade-of-service and system utilization measures) and synchronization. Subsequently, they can be used in studying the performance of the system under a variety of operational conditions, including future modifications and expansion situations.

I.1 THE STRUCTURE OF THE DATA PROCESSING NETWORK

I.1.1 System and Network Configuration and Operation

The space Shuttle avionics system contains five general purpose computers (GPCs) communicating with the avionic subsystems over serial data buses. A block diagram of the Space Shuttle Avionics system is shown in Fig. I.1.1. Four of the five GPcs are identically programmed to perform flight-critical functions, such as guidance, navigation and control. The fifth computer is programmed to perform non-flight-critical avionic functions. A block diagram of the data processing and software subsystem is shown in Fig. I.1.2.

A GPC consists of an IBM AP-101 central processing unit (CPU) and an input/output (I/O) processor (IOP). Each IOP is transformer-coupled to the buses, and can transmit or receive at a rate of 1 MHz serial digital data over each of 24 bus channels. The data buses, on the other side, are transformer-coupled to multiplexer/demultiplexer units (MDMs) and digital subsystems. The MDMs contain analog-to-digital and digital-to-analog converters. They interface with analog subsystems, such as flight control sensors and effectors (see Fig. I.1.2).

Subsystems that perform similar functions are assigned to the same data-bus group. There are seven such groups (see Fig. I.1.1). The subsystems have varying levels of redundancy at the unit level, depending on their criticality. Each unit is addressed by a command word over the bus. To prevent the loss of more than one redundant unit when one data bus fails, no two redundant units interface with the same bus.

During time-critical mission phases (i.e., recovery time less than one second), such as boost, reentry and landing, four of the five GPCs operate as a redundant set, receiving the same input data, performing the same flight-critical computations and transmitting the same output commands. In this mode of operation, efficient detection and identification of two flight-critical computer failures is provided by comparing the output commands and "voting" on the results. This involves the voting subsystem. After two failures, the remaining two computers in the set use comparison and self-test techniques to provide tolerance of a third fault. The voting mechanism thus allows a computer to transmit incorrect commands to critical subsystems for an indefinite number of cycles without having adverse effects on system operation.

The system operates as follows. Each bus within a data-bus group is assigned, under software control, to operate in either a command or a listen mode. In the command mode, data requests and commands are issued to the subsystems over the bus and data are received over the same bus. In the listen mode, data are only received on the bus.

In the flight critical sensor and control-data-bus group (two subgroups of four buses), one bus in each subgroup is assigned to operate in the command mode (in each redundant-set computer) and the remaining three are assigned to operate in the listen mode. In the inter-computer channel (ICC) data-bus group, containing five buses, one bus (in each computer) is in the command

mode and the remaining four are in the listen mode.

Data Collection. Each of the redundant subsystems is connected to a different bus. Thus, a different computer requests data from each of the subsystems and the returned data are available to all other computers in the set. The listening computers are informed that the subsystem data are available by receiving a listen command, which is issued by the command computer just prior to issuing the data request command to the subsystem. In this way, identical input data are available to each computer in the redundant set.

In noncritical phases of the mission, each of the GPCs is associated with a proper dedicated subset of subsystems. This non-redundant configuration is termed the simplex mode.

Data Output. Consider the redundant mode. Each channel of the (voting) effector subsystem is connected to a different bus of the group. Thus, a different computer transmits command data to each of the voter-effector channels. Hence, a voter-effector subsystem requires four inputs which it receives from four different computers. Since buses are interconnected to all computers, each computer can listen to the command data sent out by each of the other computers.

For inter-computer communication transfer, each computer communicates with all other computers. A computer can thus pass data to all others, request data from the other computers and perform any set of integrated tasks. No subsystem is connected to the ICC buses.

The main characteristics of the Space Shuttle orbiter avionics data processing system are summarized as follows (see Figs. I.1.1-I.1.2).

1. The avionics system provides data processing capabilities for guidance, navigation and control (GN&C); communications and tracking (C&T); displays and controls (D&C); system performance monitoring; payload management; payload handling; subsystem sequencing; and selected ground functions.
2. The system accepts input commands and/or data from the crew, on-board sensors, and external sources.
3. The system performs computations and processing. It generates output commands and data as necessary to accomplish the requirements specified for the above mentioned tasks, as well as for any required internal purposes.
4. The system is topologically structured around a central set of five general-purpose computers (GPCs) which are interconnected to the subsystems so that they may be operated in redundant groups to provide critical sources. Each computer has a memory capacity of 65,000 32-bit words. Additional storage of programs and fixed data is provided by two mass memory units, each having a data capacity of 134 megabits.
5. Data transfer between the computer center and the data users is through a data bus network. This network is composed of serial, half-duplex data channels operating at a rate of 1 megabit/sec.

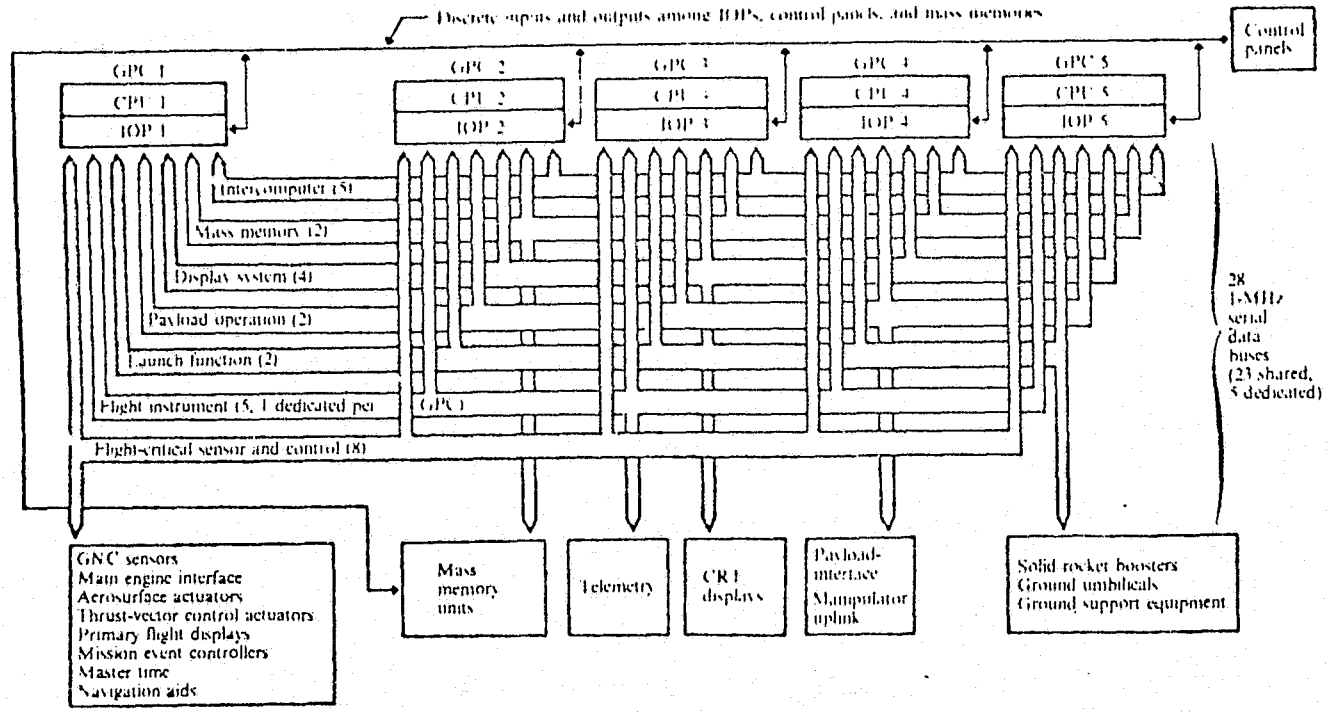

Figure I.1.1

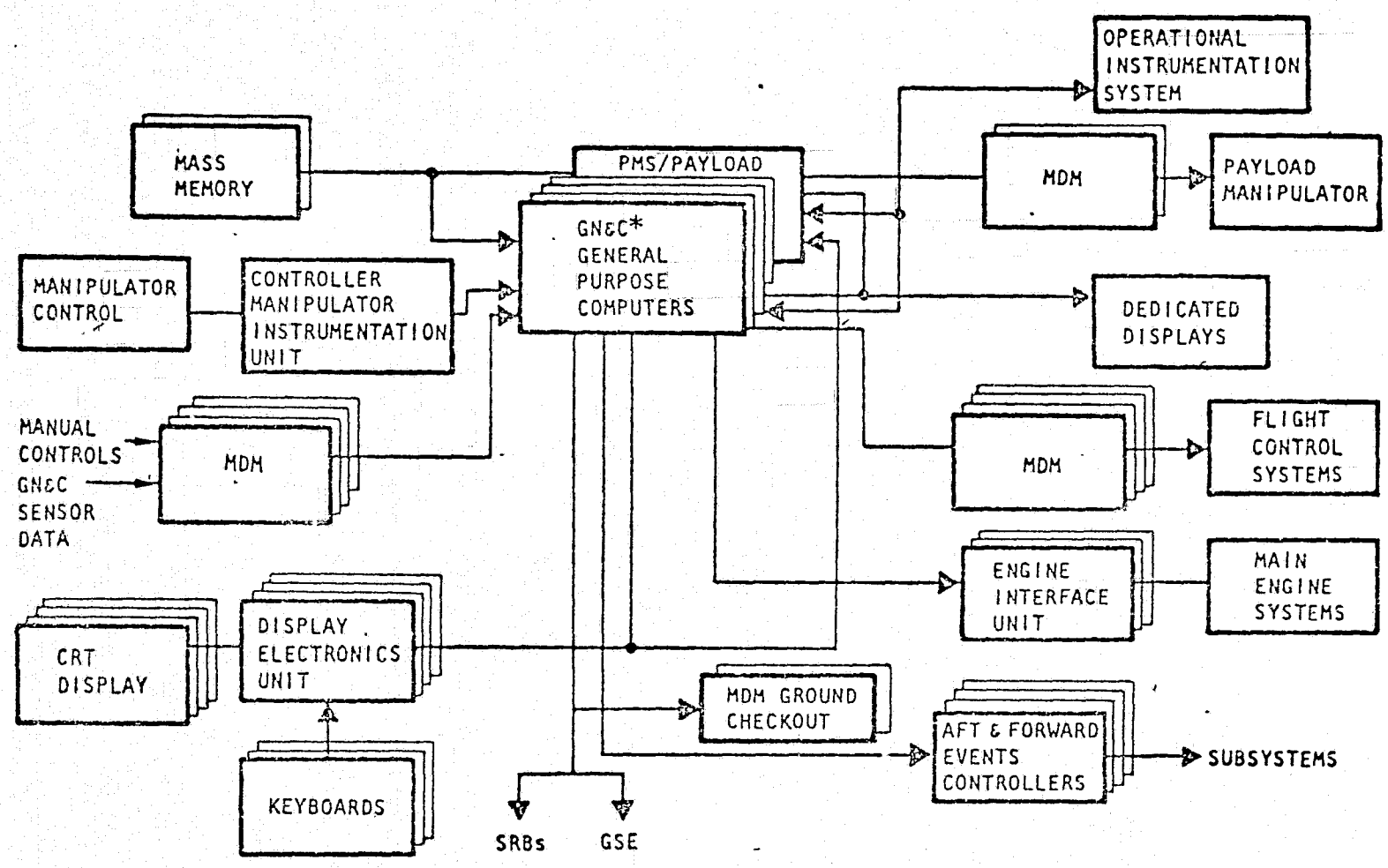

Figure I.1.2 A Block Diagram of the Data Processing and Software Subsystem.

6. Interface adaptation between the data bus network and the orbiter subsystems is accomplished by multiplexer/de-multiplexer (MDM) units. These units provide signal conversion capability, digital-to-analog (D/A) as well as analog-to-digital (A/D), and multiplexing/demultiplexing functions.
7. Engine interface units provide operational control of the main engines from GN&C commands. The units also provide main engine data for recording, telemetry or GSE.
8. Incorporated in the system are also dispaly electronics units, CRT displays, keyboards, manual controls and controller manipulator instrumentation units.

I.1.2 Characteristics of the Computer System

We have indicated in the previous section that the heart of the Space Shuttle avionics processing system is a set of five general-purpose computers (GPCs). Four of these computers can operate in a parallel redundent mode during flight critical phases of a mission. We summarize in this section the major characteristics of these computers on board the Space Shuttle orbiter.

The following are the principal characteristics of the on-board GPCs.

1. The GPCs are designed as adaptation of the IBM AP-101 computer.
2. Computer size is 0.87 cu.ft., and weight 57.9 lbs. Input power is 350 watts.
3. The computer uses transistor-transistor logic, medium and large scale integration, and multilayer interconnection boards.

4. Data flow is in parallel.
5. Both fixed point and floating point arithmetic can be used.
6. Data word length (fixed point) is equal to 16 or 32 bits.
   Data word length (floating point) is equal to 32 or 64 bits.
   Instruction word lengths are equal to 16 and 32 bits.
7. There are 154 instructions in the computer instruction repertoire.
8. The computing speed is equal to: 480×10^3 operations/sec, under fixed-point; 325×10^3 operations/sec, under floating-point.
9. The computer incorporates as special architectural features: microprogramming, a higher order language, 24 general registers and 19-level interrupt structure. As support software it contains: an assembler, a linkage editor, a simulator, a self-test program, a functional set and a compiler.
10. Memory is in the form of pluggable ferrite core modules.
    Memory capacity = 1310720 bits
    = 40960 32-bit words
    Memory access time = 0.375 μsec

The main characteristics of the computer system on-board the Space Shuttle orbiter are summarized by the following.

1. Multiple high performance computers are used to provide the total computing capacity, and system flexibility and reliability.

   During critical phases, four of the computers operate in parallel, and "voting" is used. During non-critical phases, a simplex mode is implemented. One computer is then used for GNC tasks and one for system management tasks.

2. Separate input/output (I/O) processors (IOPs) are used for information transfer and control. Each GPC consists of two separate processing units: a central processing unit (CPU), which provides the central computational capability, and an input/output processor (IOP), which performs and controls the I/O operations for the CPU.
3. Time-shared serial digital data buses are used to accomodate the data traffic among the computers and between the computers and other subsystems.

   There are 24 data buses, organized into 7 groups. The data transfer is time-division multiplexed (TDM) using pulse code modulation (PCM). Each bus operates at a clock rate of 1 Mbit/sec.
4. Microprogramming is used for both the CPU and the IOP. This allows the implementation of a comprehensive instruction repertoire.
5. Both floating-point and fixed-point arithmetic operations are provided in the CPU for easier programming and program validation.
6. A higher order language is used in the programming of the CPU to reduce software effort and yield better control. This language is designated here as HAL/S.
7. As main memory, random-access non-volatile destructive-read-out ferrite cores are used. They provide maximum reliability. Also, high capacity mass memories are used for permanent on-board off-line bulk storage to supplement the on-line random-access computer main memory. The mass memories are two identical tape units.

A functional block diagram of the GPC, showing the interconnection between the CPU and the associated IOP is shown in Fig. I.1.3. Concerning the CPU-IOP system, the following characteristics are noted.

The primary communication interface between the CPU and its IOP is provided by a 36-bit bi-directional data channel.

The main properties of the CPU have already been indicated above. We further note that the computer has a 96% fault detection capability, achieved by built-in test equipment and self-testing programs.

All data transmission among GFCs and between GPCs and the avionic subsystems is performed by the IOPs under CPU control. One IOP is associated with each CPU to provide direct and passive monitoring of data traffic.

Each IOP interfaces with the other IOPs and with the interfacing subsystems over the 24 separate serial data buses. The IOP contains a set of 24 independent processors, called Bus Control Element (BCE) processors. A 25th processor, the Master Sequence Controller (MSC) controls the operation of the 24 BCEs. These 25 processors act, in effect, as 25 digital computers and operate from software programs stored in main memory. The IOP data processing programs are independent of the CPU programs and have their own unique instruction set. Each BCE controls a Multiplexer Interface Adapter (MIA), which is connected to the serial data bus via bus computers (see Fig. I.1.3). The MIA transmits and receives information, encodes and decodes bus data, and tests for parity and proper synchronization of bits.

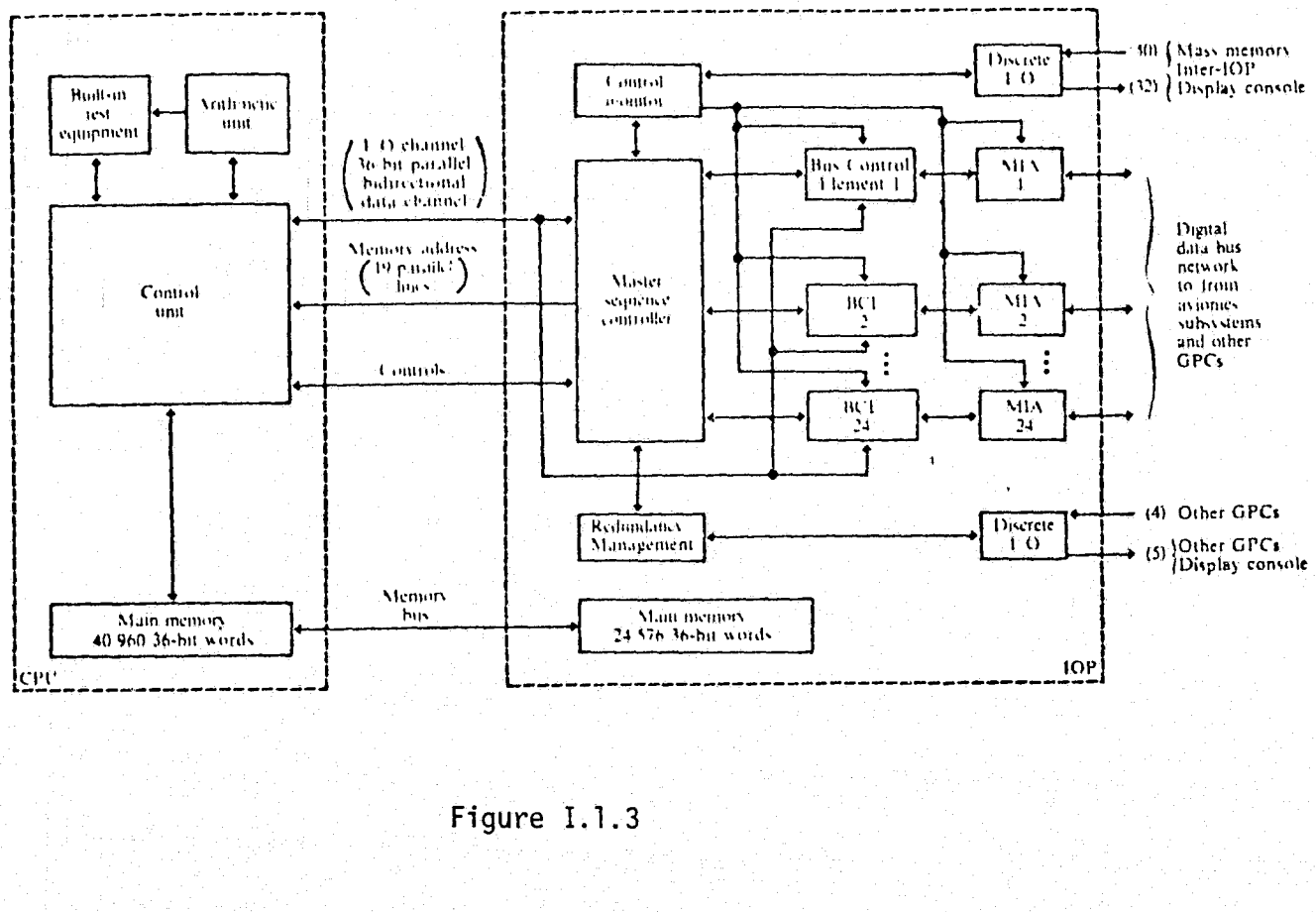

Figure I.1.3

I.2 TRAFFIC, TASK AND SUBSYSTEM MODELS AND PARAMETERS

I.2.1 The Network Components

In this section we present the main system parameters and statistical distribution functions necessary to construct an analytical model for the space Shuttle data processing system. In particular, our interest here is to construct proper queueing models that will enable the system engineer to predict and evaluate the delay-throughput performance of this computer network. The relevant set of performance measures will be presented in the next section.

In providing the parameterized models for the system components, we classify them into three categories.

1. The general purpose computers (GPCs) and the computer subsystem (complex).
2. Terminals, tasks, users and peripheral equipment.
3. The communication subnetwork.

We now consider each of these categories.

I.2.2 The Computer Subsystem

The main characteristics of the computer subsystem have already been presented in section I.1. For obtaining a global network model, we choose the following model and parameters.

The model is shown in Fig. I.2.1. The model enables us to statistically describe theprocessing services provided by the CPU and IOP, the task queueing delay characteristics, buffer overflow properties and the CPU-IOP interactions. Data and requests for service arriving at the GPC subsystem are stored in the IOP queue. Any required IOP processing is granted to the tasks

waiting at the IOP queue in accordance with the specified service ordering discipline. The latter incorporates fixed (static) priorities as well as dynamically assigned priority functions.

Subsequently, upon terminationof the desired IOP service portion, the task (or job, or message), or a request associated with it, is stored at the main queue waiting to be granted service by the CPU. The desired CPU service can involve a certain computational effort as well as memory extraction and accessing duties. The requests or data stored in the main queue are served in accordance with the underlying priority service discipline. Between various CPU service periods, the processing of the underlying task can stop so that certain IOP services or memory accesses could be completed . This is introduced into the model (see Fig. I.2.1) by allowing a CPU-IOP-CPU cycle as well as a CPU-Memory-CPU cycle. Upon termination of its service the task data output is stored at the output buffer. It is transmitted to its destination (properly controlled, as well as time-division-multiplexed by the computer IOP controls) at the proper output times.

Major parameters of interest are denoted as follows.

A_N = memory access time [sec]

C_I = IOP service rate [bits/sec]

C_C = CPU service rate [bits/sec]

M_C = Size of amin CPU memory [bits]

M_I = Size of input buffer facility [bits]

M_O = Size of output buffer facility [bits)

Some of these parameters can be random, in which case we are interested in their probability distribution functions, or just their means and variances.

The processing times requried at the CPU and IOP levels depend on the task under consideration. Considering a task of class k, distinguished by its proprity and desired response time and criticality, we are interested in the following parameters. Henceforth we identify memory processing, accessing and interruptions as I/O duties.

$S_I(k)$ = IOP total service time requried by a class k task (request, message), including memory service time [sec].

$T_I(k)$ = IOP continuous service portion required by a class k task, including memory service time [sec].

$S_C(k)$ = CPU total service time requried by a class k task [sec]

$T_C(k)$ = CPU continuous service portion required by a class k task [sec].

$K(k)$ = Number of times that a class k task required interruption in CPU processing for IOP or memory processing.

The parameters mentioned above are random variables. We are interested in their probability distributions, their means $E(\cdot)$ and variances $Var(\cdot)$. The associated means (average values) of these parameters are denoted as follows.

$$E[S_I(k)] = \overline{S}_I(k) = \tau_I(k) \text{ [sec]} \qquad \text{(I.2.2-1)}$$

$$E[T_I(k)] = \overline{T}_I(k) = \mu_I^{-1}(k) \text{ [sec]} \qquad \text{(I.2.2-2)}$$

$$E[S_C(k)] = \overline{S}_C(k) = \tau_C(k) \text{ [sec]} \qquad \text{(I.2.2-3)}$$

$$E[T_C(k)] = \overline{T}_C(k) = \mu_C^{-1}(k) \text{ [sec]} \qquad \text{(I.2.2-4)}$$

We then obtain the following relations:

$$K(k) = \frac{\tau_C(k)}{\mu_C^{-1}(k)} \qquad \text{(I.2.2-5)}$$

$$\tau_I(k) = K(k)\mu_I^{-1}(k) = \mu_I^{-1}(k)\tau_C(k)/\mu_C^{-1}(k) \; . \qquad \text{(I.2.2-6)}$$

I.2.3 Terminal, Task and User Traffic

Data traffic distribution within the Space Shuttle avionics data processing network can be associated with a number of classified "processes" or tasks. Tasks are divided into task (or message) classes in accordance with their:

proprity;

scheduled/unscheduled status

message characteristics, such as message lengths and desired response time.

Tasks can be assigned priorities on a fixed static level. Then class 1 tasks have higher priority over class 2 tasks. Priorities can also be assigned on a dynamic basis (see sections I.4-I.5 for classification of priority disciplines and the associated queueing analysis). For example, a dynamic Earlier Due Date dyanmic queueing priority discipline can also be used. Then, each task (or job, or message) is associated with dynamically changing priority level expressing the criticality of the job as well as its desired due date (response time). (See Section I.5 for details.) As a particular case, the following priority classes can be defined.

Class 1 tasks = highest priority tasks, critical.

Class 2 tasks = timely, become critical after a delay of δ_2 sec.

Class 3 tasks = timely, but become noncritical after a delay of δ_3 sec.

Class 4 tasks = timely, discarded after a delay of δ_4 sec.

Class 5 tasks = noncritical.

To implement a dynamic queueing priority service discipline, the network controller is designed to administer demand-assignment assessing and service ordering procedures.

Jobs, or tasks, are also classified in nature as being cyclic or acyclic (not cyclic) Cyclic jobs require service on a periodic basis. Acyclic tasks use the processors on an aperiodic basis.

One also distinguishes between scheduled and unscheduled tasks. Scheduled tasks can be cyclic or acyclic. They cover the following four areas.

- User interface tasks.
- System control tasks.
- Guidance, navigation and control tasks.
- System management tasks.

Tasks (jobs, or processes) are activated by either internal or external stimuli. The computer processor and the data network are assigned to tasks on a priority basis, as indicated above. Service of a task, or process, can be preempted (interrupted) by higher priority tasks. Certain tasks can be served on a non-preemptive basis. Each task is

assigned to a "service class" and given priority within the class.

In addition to representing "processes" requiring service by the Avionic DPS as tasks, one also identifies the information-bearing units called routines and messages. Routines serve as modules executed in performing a task. They can be included or shared among several different tasks. Messages are defined to be groups of data handled and transmitted within the data processing network. Messages can be declared as elements of certain tasks.

The devices associated with the Space Shuttle orbiter Avionics DPS are described as follows.

- 15 MDS (Multiplexer/Demultiplexer Units). Max. record size = 1024 bytes.
  Input/Output rates = 120 bytes/msec
  Can be shared among tasks.
- 4 DEUs (Display Electrical Units). Can be shared among tasks.
  Max. record size = 8192 bytes.
  Input rate = 120 bytes/msec.
  Output rate = 62 bytes/msec.
- 3 DDUs (Dispaly Driver Units). CAn be shared among tasks.
  Can hold an unlimited record size.
  I/O rate = 120 bytes/msec.
- 3 KBUs (Keyboard Units).
  Output rate = 1 byte/msec.
  Associated delay of 1 msec.

2 PCMMus (Pulse Code Modulation Master Units).

Can be used by all tasks.

Max. record size for each unit = 2048 bytes.

I/O rate = 120 bytes/msec.

Display data can be classified as follows.

Time critical display data. Memory resident, accessible within η_1 sec. Typically, η_1 = 1 sec.

Sequence critical data. Accessible within η_2 sec. Typically, η_2 = 2 sec. Can be resident in memory, if requried.

Noncritical data. Accessed as soon as possible. Access-time can be minimized by tape head positioning and file ordering.

In the keyboard subnetwork, a message is composed of a key-stroke or a series of keystrokes sent to the GPC system by a DEU.

The DEUs are pulled by the GPCs. Polling frequency is

f(DEU) polling times/sec

For example, in certain operational modes one sets

$$5 \text{ times/sec} \leq f(\text{DEU}) \leq 10 \text{ times/sec}$$

A given DEU receives commands from only one GPC on its bus.

DEU transactions can be very long. It is subsequently important to evaluate the probability of overflow of the associated I/O buffer.

Update data from GPCs to DEUs is transmitted at one of a number of possible rates. Typically, the rate is 2 Hz for analog data, and is equal to anyone of 1 Hz, 0.5 Hz, 0.25 Hz, 0.125 Hz for digital data.

Dedicated displays are updated regularly by the GPCs.

Dedicated control inputs are polled by the GPCs at proper polling rates.

The role of process management is to supervise the allocation of the internal computer resources and control the execution of the application processes. For that purpose, use is made of dynamic queues and tables containing the state of the internal resources.

Process control is responsible for allocation of the GPCs to application processes. This is accomplished according to (the above-mentioned) preassigned process (task) priorities, controlled by the demands of the crew, scheduled duties and conditions polled in the avionics equipment.

Scheduled processes in queues are noted to be in one of three states:

- Active state; the process controls the CPU.
- Ready state; the process is ready to utilize the CPU, but has not attained control yet.
- Wait state; time must pass until a certain event occurs or an I/O operation is completed.

I.2.4 Task and Application Process Parameters

According to the descriptions of the nature of the application processes and tasks in the previous section, the following parameters are defined. These are the major parameters used in a macroscopic performance analysis of the avionics data processing system.

Different tasks make different service demands upon the data processing network. Tasks are divided into priority (or service)

classes. GPC service times required by a class-k task have been defined in Section I.2.2. In particular, we have:

$$E[S(k)] = E[S_I(k)+S_C(k)]$$

$$= \tau(k) = \tau_I(k) + \tau_C(k) = \text{average total GPC service time required by a class k message;} \quad \text{(I.2.4-1)}$$

$$Var[S(k)] = Var[S_I(k)+S_C(k)]$$

$$= V(k) = \text{variance of the total GPC service time required by a class k message;} \quad \text{(I.2.4-2)}$$

where

$$S(k) = S_I(k) + S_C(k) = \text{total GPC service time required by a class k message.} \quad \text{(I.2.4-3)}$$

In addition to using GPC resources, a class k message might require various network and device resources. The above service times describe the overall time required by a task in directly utilizing the CPU (through $S_C(k)$) or in requiring any I/O processing (through $S_I(k)$). The local behavior and buffer overflow characteristics of each device will also be modelled.

In addition to characterizing the task service times, one also needs to statistically describe the stochastic process of task request times.

The stochastic arrival process $\{t_n(k), n=1,2,...\}$ is described as follows. The time $t_n(k)$ denotes the instant of time at which the n-th task (or, job message) of class-k signals its request for

service. This signaling can be realized by the actual arrival of request for service at the GPC, actual arrival of the proper data (or message), or any such scheduled arrival.

The interarrival times $\{T_n(k), n=1,2,\ldots\}$ are defined by

$$T_n(k) = t_n(k) - t_{n-1}(k), \quad n = 1,2,\ldots \qquad (I.2.4\text{-}4)$$

$t_0(k) \triangleq 0$. Thus, $T_n(k)$ is in general a random variable denoting the time between the arrival of the n-th class k message and the arrival of the preceeding (n-1)-st class k message. We usually assume $\{T_n(k)\}$ to be a sequence of independent identically distributed random variables. We then set:

$$\bar{T}_k(k) = E[T(k)] = \text{average interarrival time for class k messages;} \qquad (I.2.4\text{-}5)$$

$$V_T(k) = \text{Var}[T(k)] = \text{variance of the interarrival time for class k messages.} \qquad (I.2.4\text{-}6)$$

The arrival traffic associated with cyclic tasks can further be characterized as follows. The starts of the requests for service of a class k cyclic task are again governed by the stochastic arrival stream $\{t_n(k)\}$ and the associated interarrival times $\{T_n(k)\}$. However, once service has started for a certain cyclic task, the service requirement is specified by:

$$T_c(k) = \text{time between required services of a class k cyclic task} = \text{time period associated with a class k cyclic task;} \qquad (I.2.4\text{-}7)$$

$$\tau_c(k) = \text{service time of a cyclic class k task within a single associated period.} \qquad (I.2.4\text{-}8)$$

$$\bar{\tau}_C(k) = E[\tau_C(k)] = \text{mean of } \tau_C(k) \; ; \qquad \text{(I.2.4-9)}$$

$$\text{Var}[\tau_C(k)] = \text{variance of } \tau_C(k) \; . \qquad \text{(I.2.4-10)}$$

Fig. I.2.4.1 illustrates the evaluation of service times required by a class-k cyclic task.

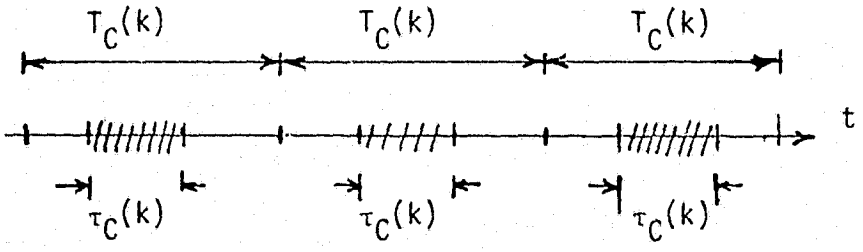

Fig. I.2.4.1.

We note that we can allow the periodic times $\tau_C(k)$, dedicated to servicing a cyclic class k task, to be identical or of random varying durations.

The arrival times $\{t_n(k)\}$ and associated interarrival times $\{T_n(k)\}$ for scheduled tasks can be regarded to be fixed deterministic values. This is observed by noting that the signals indicating request-for-service by scheduled tasks are issued at a priori known fixed instants of time.

Arrival times $\{t_n(k)\}$ and interarrival times $\{T_n(k)\}$ of requests for service of non-scheduled tasks are regarded as random variables. The mean and variance of the interarrival times, $\bar{T}(k)$ and $V_T(k)$ have been defined by (I.2.4-5) & (I.2.4-6), respectively. It can be beneficial for the advanced performance analysis to also have the interarrival time distribution function $F_{T,k}(x)$, assuming $\{T_n(k)\}$ to be a

sequence of i.i.d. random variables; thus,

$$F_{T,k}(x) = P\{T_n(k) \leq x\}, \quad x \geq 0 \qquad \text{(I.2.4-11)}$$

Unscheduled tasks are many times assumed to arrive according to a Poisson process at a rate of $\lambda(k)$ [mess./sec.]. Then, we have

$$F_{T,k}(x) = 1 - e^{-\lambda(k)x}, \quad x > 0, \qquad \text{(I.2.4-12)}$$

so that the interarrival times are exponentially distributed. Note that

$$\lambda(k) = \{E[T(k)]\}^{-1} = [\overline{T}(k)]^{-1}$$

= average number of class k task arrivals per unit time (sec)

Cyclic tasks are statistically characterized by $\{T_C(k), \tau_C(k)\}$ within each activity period. For unscheduled cyclic tasks, one can assume requests for an activity period to start at random times distributed according to a Poisson stream with intensity $\tau_C(k)$ [requests/sec].

When considering the buffer beahvior at a device, the following statistical characterizations are required.

$M^{(i)}$ = storage capacity of the buffer associated with device i.

$T_I^{(i)}$ = interarrival times of tasks (message) at device i.

$F_I^{(i)}(x) = P\{T_I^{(i)} \leq (x)\}$, $\overline{T}_I^{(i)}$, $\text{Var}(T_I^{(i)})$ = distribution, mean and variance of $T_i^{(i)}$.

$T_0^{(i)}$ = interdeparture times of tasks (messages) out of the buffer of device i.

$F_0^{(i)}(x)$, $\overline{T}_0^{(i)}$, $\text{Var}(T_0^{(i)})$ = distribution, mean and variance of $T_0^{(i)}$.

$S_I^{(i)}$ = processing time at device i.

$F_S^{(i)}(x)$, $\bar{S}_I^{(i)}$, $Var(S_I^{(i)})$ = distribution, mean and variance of $S_i^{(i)}$.

Note that task polling processes can be modelled as cyclic processes, using the characterizations presented above.

I.2.5 The Communication Subnetwork

The communication subnetwork is composed of the subsystem that provided for the transmission of information between the GPCs and the users, terminal and application devices.

For the Space Shuttle DPS, the Avionics communication subnetwork is composed of a network of bus lines. A bus line connects all computers to a certain device. The lines are used in either a command or a listen mode. In a command mode the line use is supervised and controlled by a commanding GPC to transmit or receive information. The other computers can listen. In the listen mode, a computer can only receive data over the line.

The rate of transmission of data over each bus line is 1 MHz.

To study the utilization of each bus line, we set:

$f(i)$ = rate of transmission of information over bus line (i) [bps] . (I.2.5-1)

$$f = \frac{1}{N} \sum_{i=1}^{N} f(i) = \text{average rate of data transmission over a bus line [bps]} \quad \text{(I.2.5-2)}$$

where

N = number of bus lines (connecting GPCs and devices). (I.2.5-3)

In addition, one is interested in the utilization of the ICC (inter-computer communication)lines. For which we set:

f_I = average rate of data transmission over an ICC line [bps]

(I.2.5-4)

Each bus line serves as a half-duplex communication channel. It can also be modelled as a multiplexed set of half-duplex sub-channels.

We set:

$C_L(i)$ = transmission rate over the i-th bus line [bps] (I.2.5-5)

$\Delta_L(i)$ = bit time lag over the i-th bus line [sec] (I.2.5-6)

$P_L(i)$ = probability of a bit error (due to noise, bursts, interruptions) on the i-th bus line. (I.2.5-7)

The topological structure of the communicationsubnetwork is specified by a connectivity matrix

$$C = [c_{ij}] \qquad \text{(I.2.5-8)}$$

where

$$c_{ij} = \begin{cases} 1, \text{ if node i is connected to node j} \\ 0, \text{ otherwise .} \end{cases}$$

The nodes in our network are the application devices and the processing GPCs.

In particular, we have

$$d_i = \sum_j c_{ij} = \text{degree of node i}$$

$$= \text{number of lines connected to node i.} \qquad \text{(I.2.5-9)}$$

The degree d_i of node i represents the number of lines connected to node i. For certain nodes, this number is limited by physical, performance and reliability constraints.

A routing procedure (or algorithm) needs to be specified for directing the information between the GPCs and the application devices. Involved in this algorithm is the selection of the transmission path. Related to it are the tasks of performing memory allocation, task scheduling, unit selection, element loading and I/O services.

In the Space Shuttle orbiter avionics communication subnetwork there are 27 data link buses. There are also 11 half-duplex links for interdevice communications. The data links are divided as follows.

- 5 data buses for ICC, max. transmission rate = C = 1 MHz.
- 4 data buses for display system communication, C = 1MHz.
- 8 data buses for flight critical communication, C = 1 MHz.
- 2 data buses for mission control communication, C = 1 MHz.
- 2 data buses for mass memory communication, C = 1 MHz.
- 2 ground interface buses, C = 1 MHz.
- 4 PCMMU communication buses, C = 1 MHz.
- 4 data links for communication between DEUs and DUs. C = 800 Kbps.
- 5 data links between DEUs and KBUs, C = 800 bps.
- 2 data links between PCMMUs and Instruments. C = 800 Kbps.

I.3 PERFORMANCE MEASURES

I.3.1 Computer Oriented Performance Measures

The computer complex in the Space Shuttle orbiter avionics system is the most crucial subsystem in the network, in determining the network performance. We will define this seciton the major computer oriented performance measures. In the following sections we will define user (or task) oriented and subsystem (or network) oriented performance measures.

It is important to know the extent to which we utilize the computing, processing and storing capabilities of the computer system. The following performance indices will refer to any arbitrary GPC. This is also equivalent to considering the 4 GPCs as a single computing machine for the modes in which the 4 computers are used in parallel as a redundant set.

The index of utilization of a GPC, U_C, is defined by

$$U_C = \text{relative time during which a GPC is used}$$
$$= P\{\text{a GPC is busy}\}. \qquad \text{(I.3.1-1)}$$

Note that

$$0 \leq U_C \leq 1$$

Similarly, the index of utilization of an IOP (Input/Output) processor is defined by

$$U_{IOP} = \text{relative time during which an IOP is used}$$
$$= P\{\text{a IOP is busy}\} \qquad \text{(I.3.1-2)}$$

Note that $0 \leq U_{IOP} \leq 1$.

The index of utilization of a CPU is given as

U_{CPU} = relative time during which a CPU is used

= P{a CPU is busy}. (I.3.1-3)

Also, $0 \leq U_{CPU} \leq 1$.

The overall GPC system is composed of the CPU, IOP and associated memory and storage facilities. One can thus define a GPC to be busy if either its CPU or its IOP, or both, are busy (i.e., used for processing, computing or active storing). Then, we will have

$$1 - U_C = (1-U_{IOP})(1-U_{CPU}) \quad \text{(I.3.1-4)}$$

so that

$$U_C = 1 - (1-U_{IOP})(1-U_{CPU})$$
$$= U_{CPU} + U_{IOP} - U_{IOP}U_{CPU} \quad \text{(I.3.1-5)}$$

It is also many times of interest to find the statistical characteristics governing the use of GPC. We identify alternating idle periods and busy periods in observing the use of CPU, IOP and the GPC buffers. We then define:

$\overline{B}_{CPU}$, $Var(B_{CPU})$ = mean and variance of the busy-period duration B_{CPU} for the CPU (I.3.1-6)

$\overline{I}_{CPU}$, $Var(I_{CPU})$ = mean and variance of the idle-period duration I_{CPU} for the CPU (I.3.1-7)

$\overline{B}_{IOP}$, $Var(B_{IOP})$ = mean and variance of the IOP busy-period (I.3.1-8)

$\overline{I}_{IOP}$, $Var(I_{IOP})$ = mean and variance of the IOP idle-period (I.3.1-9)

$\overline{B}_C$, $Var(B_C)$ = mean and variance of the GPC busy-period (I.3.1-10)

$\bar{I}_C$, $Var(I_C)$ = mean and variance of the GPC idle-period (I.3.1-11)

It is important to also measure the utilization of the memory and storage devices. For that purpose, the following performance indices are defined.

UM_C = index of utilization of the GPC

= memory average fractional part of the GPC memory which is not used. (I.3.1-12)

UM_I = index of utilization of the GPC input buffer (I.3.1-13)

POF_I = probability of overflow of the GPC input buffer (I.3.1-14)

UM_O = index of utilization of the GPC output buffer (I.3.1-15)

POF_O = probability of overflow of the GPC output buffer (I.3.1-16)

The GPC throughput index is used to assess the average amount of data processed, and tasks performed, by the GPC per unit time. Thus:

TH_C = the GPC throughput

= average number of bits per sec served by the GPC (I.3.1-17)

We can also consider the number of tasks per unit time performed by the computer:

TTH_C = the GPC task (job, message) throughput

= average number of tasks (jobs, messages) processed by the GPC (or computer complex) per sec (I.3.1-18)

I.3.2 User Oriented Performance Measures

The major index of performance associated with a user or a task (job, message) is the associated task time delay.

Tasks (jobs or messages) are classified into classes (as detailed in Section I.2) in accordance with their priorities, criticality and required time delays.

The response-time or time delay of a class-k task is denoted by

$$D(k) = \text{time-delay, response-time of a class k task (message, job)} \qquad \text{(I.3.2-1)}$$

The response-time D(k) is the period of time measured from the instant of the class-k task records its request for service to the instant its service has been completed.

We also set:

$$W(k) = \text{waiting-time of a class k task} = \text{time from the instant the task request is recorded to the instant its service starts} \qquad \text{(I.3.2-2)}$$

Thus, W(k) denotes the time duration that a class task is delayed until its processing has started.

The processing time required by a class k task has been defined (see (I.2.4.3)) as S(k). We then have that

$$D(k) = W(k) + S(k) \text{ [sec]} \qquad \text{(I.3.2-3)}$$

The time-delay and waiting-time functions are random variables. We are generally interested in their distributions:

$$F_{D,k}(x) = P\{D(k) \leq x\} \quad , x \geq 0; \qquad \text{(I.3.2-4)}$$

$$F_{W,k}(x) = P\{W(k) \leq x\} \quad , x \geq 0 . \qquad \text{(I.3.2-5)}$$

In particular, it is of interest to use as a performance measure the user average time-delay. We set:

$\overline{D}(k) = E[D(k)]$ = average task k time-delay (response time); (I.3.2-6)

$\overline{W}(k) = E[W(k)]$ = average task k waiting time (I.3.2-7)

Since

$\overline{S}(k)$ = average processing time required by a class k task,

we have

$$\overline{D}(k) = \overline{W}(k) + \overline{S}(k) \ . \qquad \text{(I.3.2-8)}$$

It is also important in many cases to evaluate the variances of the task delay and waiting times:

$$\text{Var}[W(k)],\ \text{Var}[D(k)]; \qquad \text{(I.3.2-9)}$$

$$\text{Var}[D(k)] = \text{Var}[W(k)] + \text{Var}[S(k)] \ . \qquad \text{(I.3.2-10)}$$

The standard deviation of the class-k task response-time is then given by

$$\sigma(k) = \sqrt{\text{Var}\ D(k)} \ . \qquad \text{(I.3.2-11)}$$

In measuring the peak task response-time, one is interested in the probability

$$P\{|D(k) - \overline{D}(k)|\} > \alpha \quad , \qquad \text{(I.3.2-12)}$$

expressing the probability (fraction of time) that the response time deviates from its average value by α. By Chebychev's inequality, we conclude that

$$P\{|D(k) - \overline{D}(k)| > 3\sigma(k)\} \leq \frac{1}{9} \approx 11\% \qquad \text{(I.3.2-13)}$$

Therefore, we can estimate the peak delay of a class k task by setting

$$\therefore \quad \overline{D}_p(k) = \overline{D}(k) + 3\sigma(k) \ . \qquad \text{(I.3.2-14)}$$

Relation (I.3.2-13) indicates that more than 89% of the time the delay D(k) will be lower than this $\overline{D}_p$ value.

Other user related performance measures can be defined in relation to specific modes of operation. In certain cases, some tasks are rejected for processing. We then set:

$P_R(k)$ = probability that a class k task is rejected. (I.3.2-15)

Certain devices, or terminals (users) experience local queueing phenomena. Considering device i, one then defines:

$U_D(i)$ = index of utilization of the device i buffer; (I.3.2-16)

$\overline{M}_D(i)$ = average occupancy of the device i buffer; (I.3.2-17)

POF(i) = probability of overflow of the device i buffer; (I.3.2-18)

User related reliability measures are of prime importance as well. These will be detailed in the section on network reliability. In particular, it is of interest to specify and compute the following measures:

L(k) = probability of loss of a class k message . (I.3.2-19)

$L_D(k)$ = probability that class k message (job, task) does not receive service within D sec. (I.3.2-20)

I.3.3 System and Network Related Performance Indices

The reliability issue of the topological structure of the network gives rise to a number of invulnerability measures. In particular, one defines:

K(i) = minimal number of line failures that disconnect device i (or application process i) from the computer complex; (I.3.3-1)

$P_K(i)$ = probability that device i, or application process i, will be disconnected from the processing resources (due to line failures, terminal failures or GPC failures). (I.3.3-2)

An overall network throughput measure is

TTH = average number of tasks processed by the system per unit time. (I.3.3-3)

We can then write

$$TTH = \sum_k TTH(k). \quad \text{(I.3.3-4)}$$

The network delay measures are specified by the values $\{\bar{D}(k)\}$, $\{\bar{D}(k)+3\sigma(k)\}$. Indices of utilization of the GPC memory and buffers and the device buffers have been defined above.

Performance indices indicating the sensitivity of the network operation to fluctuations in traffic are important. For that purpose, we set

$\Delta\bar{D}(k)$ = change in the average class k message delay as a result of the increase of the overall traffic rate according to $\{\Delta\lambda(k)$ [mess./sec]$\}$.

ΔTTH = change in the network throughput with the Δ increase of intensity of task demands.

Also of importance are measures indicating the growth capability of the network. In particular, we set:

$\Delta\overline{M}_B(D_k), \Delta\overline{M}_C(D_k)$ = average growth allowed in occupancy of computer buffer (B), or memory (C), attaining average task delays not higher than $\{D_k\}$.

$\Delta U_C(D_k), \Delta U(i,D_k)$ = average growth allowed in index of utilization of GPC elements (UPC, IOP, buffers), or device i elements, causing message delays not higher than $\{D_k\}$.

The following sections will present proper queueing models to be employed in analyzing the Space Shuttle DPS. We will also present performance analysis results for such models. The network and computer complex designer will then be able to apply the proper model to the underlying subsystem he is analyzing. He will subsequently be able to compute the set of relevant performance measures indicated above. In particular, note the following main families of performance indices that we defined above, and will compute in the following sections.

- Task (job, message) response times (queueing and service time delays).
- System Throughput.
- System indices of utilization.
- Reliability measures.
- Performance sensitivity measures.
- Network growth measures.

I.4 TIME-SHARING QUEUEING MODELS

I.4.1 Time-Shared Single Processor Systems

We consider a queueing model for a system wheren the serving resource is modelled as a single server queue. This resource can model the GPC of the Shuttle orbiter avionics systems. The service provided by the latter includes the relevant GPC CPU and IOP processing functions. Demands are made upon this single server processor by the arriving messages or requests. Due to the finite resources available to the server, and its finite processing rates, arriving messages will have to be queued at a buffer before they can be processed. A scheduling algorithm needs then to be devised to control the assignment of service resources to the arriving messages and demands.

We consider in this section such scheduling algorithms that use the service facility on time-shared basis.

The general structure of the queueing model is shown in Fig. I.4 1

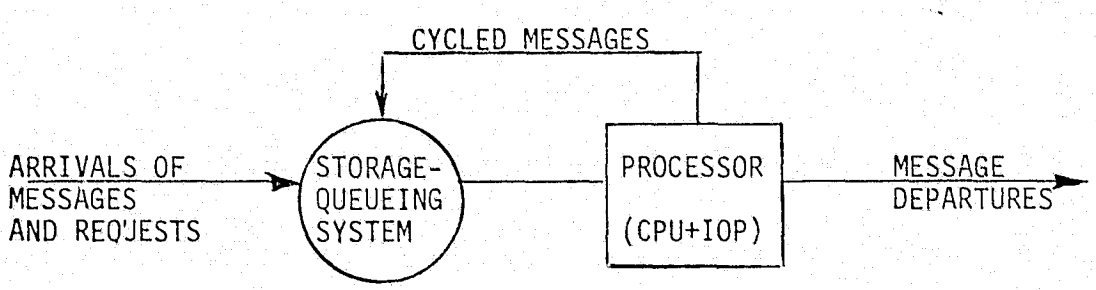

Figure I.4 1

In the typical time-shared system, one generally wishes to attain a message average queueing delay (response time) which is proportional to the average message length. Thus, short

messages expect to experience short waiting-times, while long messages are prescribed longer time delays. This is achieved by the feedback queueing model shown in Fig. I.4.1-1.

In this time-sharing system, the server (GPC) allows a message to stay in service (be processed) only for a certain time period, called quantum. The quantum duration may vary, and it can depend upon the state of the system, the message priority and the message past processing record. If the processing time required by the message or request is not satisfied by the end of the quantum service period, the message is returned to the queueing (storage) system, where it joins the queue of messages waiting for service. Otherwise, the processing required by the message has terminated and it leaves the GPC to its destination.

I.4.2 Traffic and Performance Parameters

We need to statistically describe the stream of message arrivals at the server, and the service (processing) demands made by each message.

For that purpose, we generally assume message interarrival times to be independent identically distributed random variables. The message interarrival time distribution is set equal to

$$A(t) = P\{\text{interarrival time} \leq t\} \quad . \qquad \text{(I.4.2-1)}$$

Message service times, or required overall GPC (processor) times, are generally assumed to be independent identically distributed random variables, for any specific class (or priority group) of message. We then set the message service time distribution to be

$$B(t) = P\{\text{message service time} \leq t\} \quad . \qquad \text{(I.4.2-2)}$$

It is common to assume that messages arrive according to a Poisson process with intensity λ [mess./sec]. This amounts to assuming an exponential distribution for the interarrival time:

$$A(t) = 1 - e^{-\lambda t} \quad , \quad t \geq 0 \qquad \text{(I.4.2-3)}$$

A Poisson arrival streams models a complete random stream of arrivals (in that the interarrival durations follow a memoryless distribution).

It is also many times convenient, to simplify analytical studies, to assume the required message service time to be exponentially distributed. Then

$$B(t) = 1 - e^{-\mu t} \quad , \quad t \geq 0 \ , \qquad \text{(I.4.2-4)}$$

and we set the

$$\text{Average Message Service Time} = \mu^{-1} \quad \text{[sec/mess.]} \ . \qquad \text{(I.4.2-5)}$$

In a time-sharing system, the quantum service provided by the processor is usually set equal to a constant Δ, or is defined as Δ_{pn} to depend upon the message priority class p and upon its number (n) of prior entries into service. Also included in this quantum duration is the swap time period, spent in transferring messages between the queueing and service facilities.

The main performance measure used in this section is the message average time delay (response time) D. It represents the average overall time spent in the queueing system by the server. The average time spent by a message is waiting at the queueing facility is denoted by W. The average message service time is S. We clearly have

$$D = W + S \qquad \text{(I.4.2-6)}$$

As a major objective of this feedback system is to attain a message time delay proportional to the message length T, $D \alpha T$, we can represent explicitly the delay and waiting time measures for a message as functions of its required service time T, and denote then by D(T) and W(T), respectively. We have

$$D(T) = W(T) + T \qquad \text{(I.4.2-7)}$$

In the following, we describe certain useful scheduling algorithms for time-sharing systems, and indicate their performance characteristics.

We assume messages to arrive according to a Poisson process with intensity λ [mess./sec].

I.4.3 Batch Processing: First-Come First-Served

The structure of the basic queueing system, where no feedback is employed is shown in Fig. I.4-2.

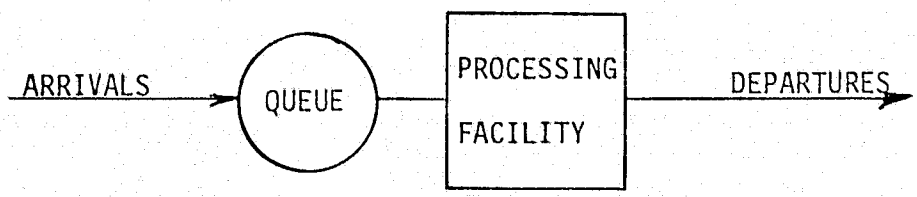

Fig. I.4-2.

Messages arriving at the system are stored in a queue. They are served by the single server (processor) on a first-come first-served basis. Once a message is accepted into service, it is allowed to complete its processing. (The quantum is thus of infinite duration.) The average message response time D(S) is given by the well-known Pollaczeck-Khintchine formula:

$$D(T) = \frac{\lambda \overline{S^2}}{2(1-\rho)} + T, \quad \text{for } \rho < 1, \tag{I.4.3-1}$$

where

$$\rho = \lambda S = \text{traffic intensity parameter;} \tag{I.4.3-2}$$

$$S = \int_0^\infty t\,dB(t) = \text{average message processing time;} \tag{I.4.3-3}$$

$$\overline{S^2} = \int_0^\infty t^2 dB(t) = \text{second moment of message processing time.} \tag{I.4.3-4}$$

Note that

$$\overline{S^2} = \sigma^2 + S^2 \ , \tag{I.4.3-5}$$

where σ is the standard deviation of the message service time.

The traffic intensity parameter ρ yields the ratio

$$\rho = \frac{\text{average message processing time}}{\text{average message interarrival time}} \tag{I.4.3-6}$$

It is a measure of congestion in the system. We obtain

$$W = D = \infty \ , \quad \text{if } \rho \geq 1 \ , \tag{I.4.3-7}$$

so that arbitrarily high time delays are experienced, as the system evolves in time, by messages if $\rho \geq 1$. Hence, we are interested in operating the system such that $0 \leq \rho < 1$, and finite queueing delays result.

We note that the average message response time depends only on the first two moments of the required message processing time, and not on its distribution.

The message average waiting time $W(T) = D(T)-T$, is given by

$$W(T) = \frac{\lambda \overline{S^2}}{2(1-\rho)} \quad , \quad \text{for } \rho < 1 \ . \qquad \text{(I.4.3-8)}$$

For the special case, where message processing times follow exponential distribution (2.4), we have

$$\overline{S} = \frac{1}{\mu} \ , \quad \overline{S^2} = 2(1/\mu)^2 \quad , \qquad \text{(I.4.3-9)}$$

so that the message waiting time is given by

$$W = \frac{\lambda/\mu}{1-\rho} \quad , \text{ where } \rho = \frac{\lambda}{\mu} < 1 \quad . \qquad \text{(I.4.3-10)}$$

Thus, for this (FCFs) model, the waiting time W is independent of the message required processing time T.

As noted in Fig. I.4-3, the message waiting time function W becomes a very sensitive function of ρ as ρ approaches 1. Thus, the messsage queueing delay increases very fast as the system's congestion approaches its saturation value. One should therefore design the system so that it avoids the traffic intensity region close to saturation, i.e., close to ρ=1.

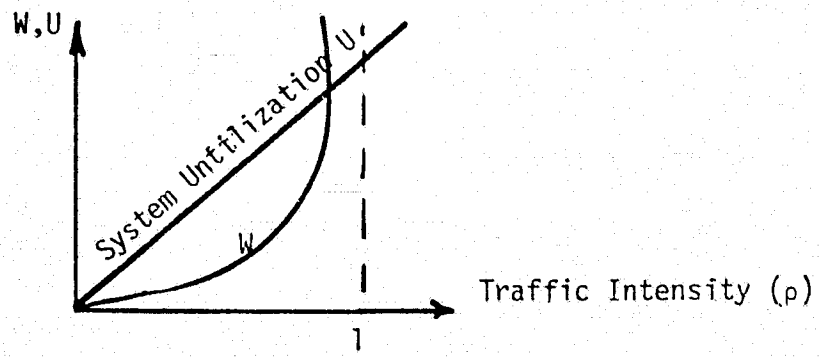

Figure I.4-3.

The average queue-size parameter $\overline{X}$, describing the average number of messages in the system, is given for the latter queueing system by

$$\overline{X} = \frac{\rho}{1-\rho} \quad , \text{ for } \rho < 1 \qquad \text{(I.4.3-11)}$$

Note that for $\rho = 0.8$, only an average of 4 messages are in the processing system (queued or being processed), while for $\rho = 0.9$ and 0.99, the average queue size is equal to 9 and 99, respectively.

We also have that

$$P\{\text{system is empty}\} = 1-\rho \ , \quad \text{for } \rho < 1 \ , \qquad \text{(I.4.3-12)}$$

so that

$$U = P\{\text{system is busy}\} = \rho \ , \quad \text{for } \rho < 1 \ . \qquad \text{(I.4.3-13)}$$

Thus, ρ serves as a measure of system utilization. For $\rho < 1$, the processing system is kept busy a fraction ρ of time. For $\rho = 0.8$ and 0.9 the processor is busy 80% and 90% of the time, respectively.

Clearly, one must compromise between having high enough a processor utilization factor and low enough message response times. The system utilization index $U = \rho$ is also shown in Fig. I.4-3.

I.4.4 Round-Robin Processing

In a Round-Robin (RR) processing, the processing (GPC) facility serves each message for a fixed quantum period Δ. Newly arrived messages join the end of the queue. When they arrive at the end of the queue they are sent into the processing facility where they are served for a period of Δ sec. Then, if their service demand is fully satisfied, they leave the system. Otherwise they are cycled back to the end of the queue, starting again the same queueing-service process. A RR system structure is illustrated by Fig. I.4-4.

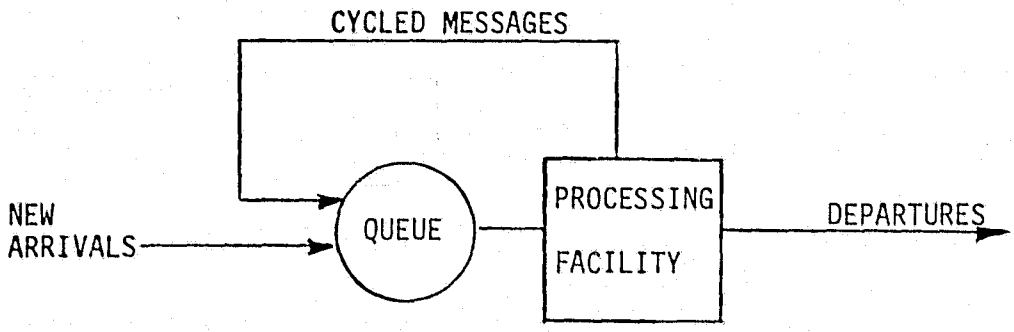

Figure I.4-4.

The RR service discipline can also be regarded as a processor sharing service procedure. To explain this notion, we note that when there are n messages in the system, and if Δ is small, each message is in fact processed (served) by the processing facility at a rate of $\frac{1}{n}$ sec/sec. Thus, we can regard the processor as shared among the various messages on an equal basis.

For a round-robin system with arbitrarily small Δ value, the average message delay D(T) is given by

$$D(T) = \frac{T}{1-\rho} \quad , \quad \text{for } \rho = \lambda S < 1, \qquad \text{(I.4.4-1)}$$

where T is the required message processing time. The average message waiting time is then equal to

$$W(T) = \frac{\rho T}{1-\rho} \quad , \quad \text{for } \rho = \lambda S < 1 \ . \qquad \text{(I.4.4-2)}$$

Thus, the RR system yields a message response time which is linearly dependent on the required message processing time T.

For exponentially distributed message service times, we can note that messages requiring shorter (longer) processing times than the average one will experience shorter (longer) response times in

a round-robin system than in a first-come first-served system.

I.4.5 Round-Robin with Priorities

We divide the arriving messages into P priority classes. A p-priority message is a message which belongs to priority group p, where p is an integer in {1,2,...,P}. A p-priority message is considered to have higher priority than a q-priority message if $p < q$.

We assume P streams of message arrivals at the single server queueing system. The stream of p-priority messages is taken to be a Poisson process with intensity λ_p [mess./sec].

Assume p-priority messages to have exponentially distributed processing (service) times with mean $\frac{1}{\mu_p}$ [sec/mess.].

A p-priority message is assigned an r_p fraction of the processing time. We can choose r_p as desired, setting higher r_p values for higher priority (lower p) messages.

For example, let f_p be an arbitrarily chosen function that sets higher values to higher priority (lower p) messages. When there are x_i messages at the system from the i-th group, i=1,2,...,P, we set the fraction r_p of processing time dedicated to the p-priority customer to be

$$r_p = \frac{f_p}{\sum_{i=1}^{P} f_i x_i} \qquad \text{(I.4.5-1)}$$

Thus, we have specified a processor sharing system where the share of the processor assigned to each message depends upon its priority group.

The average delay $D_p(T)$ for a p-priority message is then

$$D_p(T) = \frac{T}{1-\rho}\left[1 + \sum_{i=1}^{p}\left(\frac{f_i}{f_p} - 1\right)\rho_i\right] , \qquad (I.4.5\text{-}2)$$

where

$$\rho_i = \frac{\lambda_i}{\mu_i} < 1, \quad \rho = \sum_{i=1}^{p} \rho_i \ . \qquad (I.4.5\text{-}3)$$

Thus, the message response time again depends linearly upon the message service time T, as for the round-robin system. But, in addition, we obtain the message response time to depend upon the message priority class.

By properly choosing the discrimination function f_p, we can separate as we wish between the response-time vs ρ curves of the various priority classes. Typical curves for the message waiting-time functions W_p are shown in Fig. I.4-5.

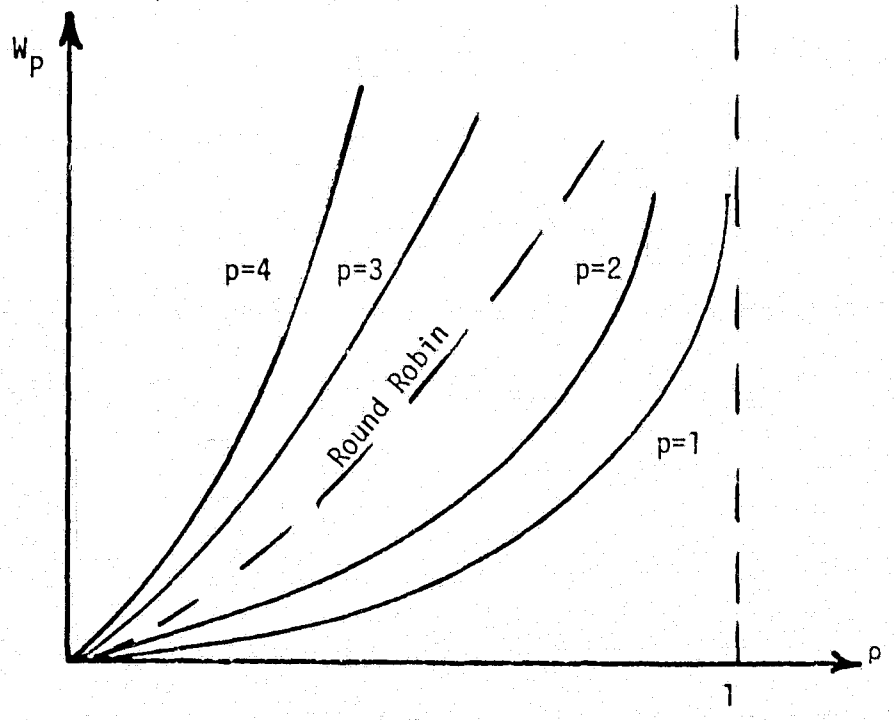

Figure I.4-5.

I.4.6 A Round-Robin Scheme with Time-Varying Priorities

We can assign a time-varying priority index to each message, depending on whether he is being in processing or stored in the queue. Thus, the priority of a message is set to increase linearly at a rate α whenever it is waiting in the queueing facility. His priority is, on the other hand, set to increase at a lower rate β, where

$$\alpha \geq \beta \geq 0 ,$$

when it is in the service (processing) facility.

Service is provided to all messages in the system which presently have the highest priority. When more than one message have the present highest priority value, all the latter are served in a round-robin fashion, thus sharing the processor resrouces.

An entering message will then increase its priority at a higher rate than those currently served. Eventually, this message catches-up with those being processed. Then it is entered into the service facility and remains there until its service demand is satisfied.

The average message delay D(T) in this system is given by

$$D(T) = \frac{1/\mu}{1-\rho} + \frac{T - \frac{1}{\mu}}{1 - \rho[1 - \frac{\beta}{\alpha}]} , \qquad \text{(I.4.6-1)}$$

where

$$\rho = \lambda/\mu < 1 , \qquad \text{(I.4.6-2)}$$

λ is the intensity of the Poisson message arrivals, and message service times are exponentially distributed with an average message processing time of

$$S = \frac{1}{\mu} \text{ [sec/mess.]} .$$

The average message waiting time is

$$W(T) = D(T) - T \qquad \text{(I.4.6-3)}$$

The dependence of the message waiting time W(T) on the requried processing time T is shown in Fig. I.4-6, where the ratio β/α is a parameter.

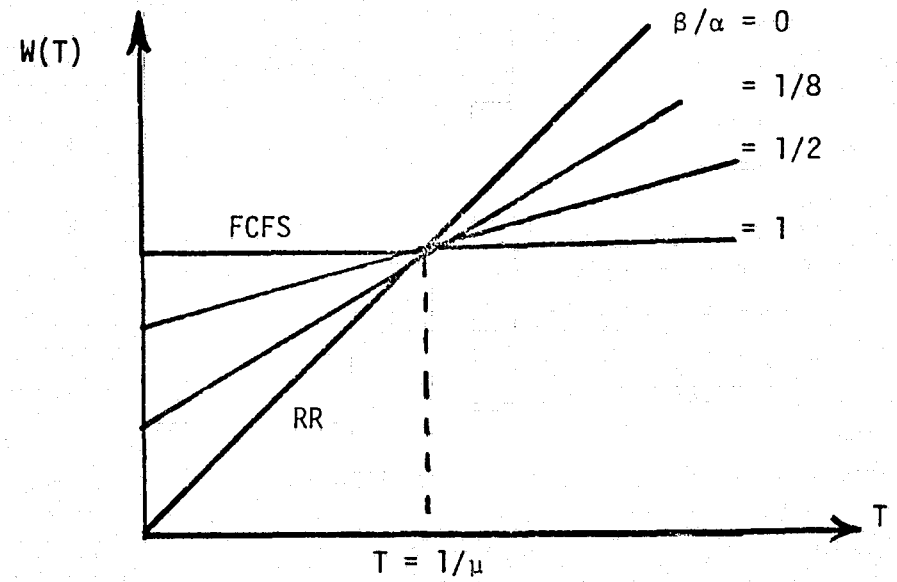

Figure I.4-6.

We note that a message which requires a processing time T equal to the average one, $T = 1/\mu$, will experience the same response time under any β/α value.

When $\beta/\alpha = 1$, the oldest message in the system captures the processor and uses it for itself alone. Hence, we obtain a FCFS queueing scheme.

When $\beta=0$, we obtain a round robin scheme, since a message does not gain priority while being processes.

Changing β/α between 0 and 1 we obtain response time curves that vary continuously between those of a FCFS system and a RR scheme.

I.4.7 Foreground-Background Processing Schemes

In a foreground-background service scheme, we have two queues. A newly arrived message joints the first queue. It receives there its first quantum of service Δ_1, being served on a FCFS basis. Thereafter, the message joins the end of a second queue. From then on the message can join only the second queue. The processing facility always serves first the messages in the first queue, called also the foreground messages (or jobs, tasks). When the first queue is empty, the processor turns to serve the background messages queued in the second queue.

The generalized foreground-background (FB) scheduling scheme described next is structured so that service is always given to that message which has so far received the least service of all.

A new message which finds the system empty is given the full attention of the processor. It is served at a rate of 1 sec/sec. If prior to thetermination of its service, a new message arrives, the processor gives its full service to the second message. This continues until the second message has received the same service time as the first one. Subsequently, if there are no new arrivals and these messages have not yet departed, the processor is shared among these messages, yielding each service rate of 1/2 sec/sec; and so on. Thus, the processor always serves those messages that have so far received the least service.

The average message response time D(T) is given by

$$D(T) = \frac{A(T) + T}{1-\rho_T} \qquad \text{(I.4.7-1)}$$

where

$$A(T) = \frac{\lambda \overline{S(T)^2}}{2(1-\rho_T)} \quad , \tag{I.4.7-2}$$

$$\overline{S(T)^2} = \int_0^{T_1} x^2 dB(x) + T^2[1-B(T)] \quad , \tag{I.4.7-3}$$

$$\overline{S(T)} = \int_0^{T} x dB(x) + T[1-B(T)] \tag{I.4.7-4}$$

$$\rho_T = \lambda \overline{S(T)} < 1 \quad , \tag{I.4.7-5}$$

where $B(x)$ is the distribution function of the message processing time. We note that

$$\overline{S(\infty)} = S, \quad \overline{S(\infty)^2} = \overline{S^2} \quad , \quad \rho_\infty = \rho \quad , \tag{I.4.7-6}$$

and

$$A(\infty) = W(FCFS) \tag{I.4.7-7}$$

where $W(FCFS)$ is the message waiting time in a FCFS queueing system.

A typical curve of $D(T)$, for exponential service times, is shown in Fig. I.4-7.

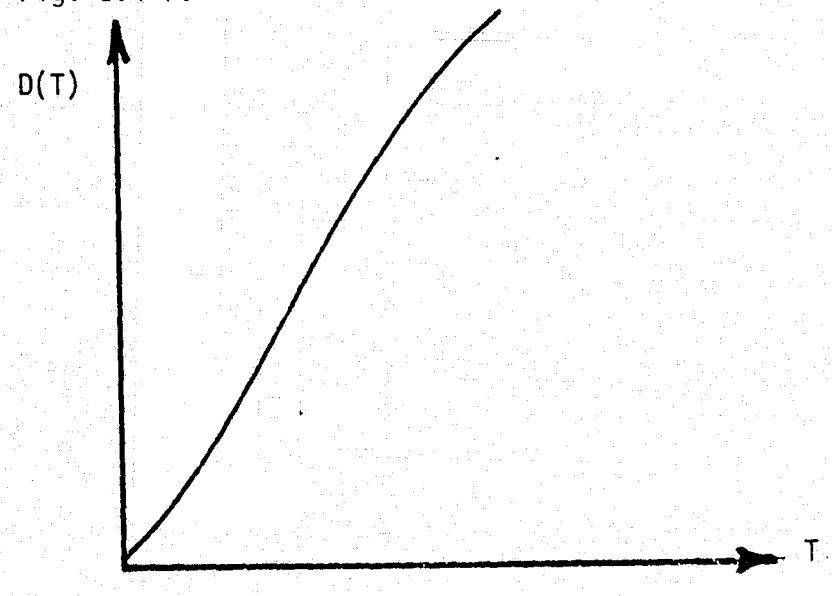

Figure I.4-7

One computes the slop of D(T) to be equal to L at T=0 and to $1/1-\rho$ at $T = \infty$. Thus, a message with a very short required processing time is given here a service rate close to unity. On the other hand, messages requiring very long processing times has to wait until all the messages arriving during its requried service time are first processes; thus experiencing a service rate equal to that given to it in a RR scheme.

I.4.8 Multilevel Processor Sharing Schemes

A family of multilevel processor sharing service disciplines can be defined by dividing the message (or job, task, process) processing time into the $\{a_i\}$ values:

$$0 = a_0 < a_1 < a_2 <\ldots< a_N < a_{N+1} = \infty.$$

We now define N+1 scheduling procedure. The i-th procedure $(SP)_i$ is applied when the message has been received the service value of x in the interval

$$a_{i-1} \leq x < a_i \quad , \quad i = 1,2,\ldots,N+1 \ .$$

We can set $(SP)_i$ to be either FCFS, FB or RR. Also, between these intervals messages are treated as foreground-background jobs, so that the processor gives its complete attention to messages in the lowest level nonempty queue.

As a FB discipline is used between level, one can observe that the message response time depends only on the discipline used when it departs from the system, after receiving its complete processing requirement.

Subsequently when a message departs at the i-th level,

receiving there FCFS service, his delay time is given by

$$D(T) = \frac{A(a_i) + T}{1-\rho_{a_i}-1}$$

where $A(x)$ is given by (I.4.7-2) and ρ_x by (I.4.7-5).

When the message last level i uses an FB discipline, D(T) is given by the FB formulas, assuming the entire level below to use FB procedure.

For exponentially distributed message processing times, one can note the response curves D(T) for FCFS multilevels or RR multilevels to be close to the response curve obtained when a single FB service discipline is used.

One can properly choose the various levels and associated disciplines so that a D(T) curve with certain desired characteristics are obtained for the underlying processing system.

I.4.9 Comparing the Performances of the Time-Sharing Schemes

The message delays experienced under the various time-sharing schemes presented above can be compared as follows.

For messages that require very short processing times the foreground-background (FB) service discipline yields the shortest response times.. Comparable performance is exhibited then also by a round-robin (RR) scheme.

For messages that require long processing times, the first-come first-served (FCFS) service disciplines yields the lowest response time values. This is also the case when medium-valued required service times are involved.

The round-robin scheme with time varying priorities, also called Selfish Round Robin (SRR) scheme, as well as the Multi-level (ML)

scheme, yield D(T) curves that are between those of the FCFS and FB ones. Figure I.4-8 illustrates the typical situation.

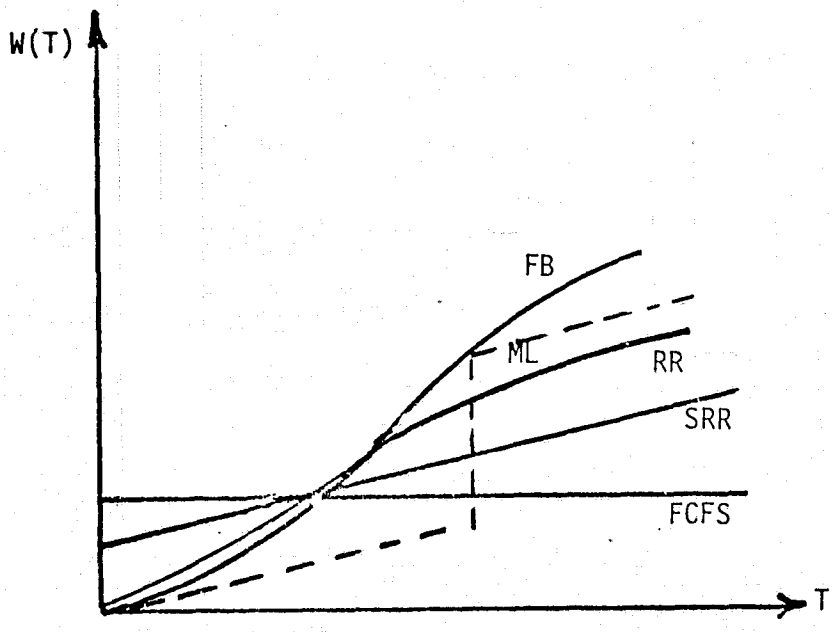

Figure I.4-8

In designing the time-shared processing part of our computer system, we can thus attain the proper response time D(T) vs requried processing time (T) curve, by choosing the proper multi-level (ML) scheme, or jsut a FCFS, RR, SRR or FB scheme. The optimal choice can be made based on the presented results, for each traffic-message environment under consideration.

I.5 PRIORITY QUEUEING MODELS

I.5.1 On Service Disciplines

Messages or requests for service arriving at the central processing (GPC) system are queued (stored) in a buffer until the processor is ready to serve them. These messages need to be properly ordered for service. This ordering follows the service discipline, or scheduling algorithm, governing the operation of the underlying queueing system.

A multitude of service disciplines can be defined and implemented in our data processing system. Different disciplines will be required at different times, while different jobs and tasks require service. It is thus of importance to implement a dynamic (flexible) scheduling rule.

In this section we classify and discuss some of the priority service models of importance and relevance to the Space Shuttle orbiter avionics system under consideration.

In designing a scheduling algorithm, one can assume the a priori distribution of priorities among the various messages (or tasks), according to their desired response time and measure of importance or urgency. In turn, one wishes to dynamically modify the order of service of messages in the system in accordance with the state of the system, so that a proper performance measure is optimized. Such a performance measure involves the satisfaction of the required response times by the various messages, in accordance with their class, urgency and statistical characteristics.

I.5.2 Scheduling Algroithms for Time-Shared Processing Systems

We have described in Section I.4 a multitude of scheduling

algorithms for time shared processing systems. We have also given there the associated response time functions and compared the performance characteristics of the various schemes.

In these time-sharing systems, the processing center has been time-shared among the messages. A single processing unit has been assumed. The following disciplines have been noted.

First-come first-served (FSFS) service discipline. Messages are served in order of arrival. Thus, messages are queued at the storage facility in the order of their arrival. When the processor becomes free, the message at the head of the queue is accepted for processing.

Round-robin (RR) service discipline. The processing system is time-shared among the messages in the system on an equal basis. Thus, if n messages (tasks) are in the system (requiring processing), each message is processed at a rate of 1/n sec/sec.

Round-Robin with Priorities. The processor service time is shared among the messages (jobs) in the system in accordance with the message priority class. Thus, messages which belong to a higher priority class are assigned a higher service rate.

Round-Robin with Time-Varying Priorities. Messages that are currently being processed are assigned a lower shared processing rate than messages that have just arrived.

Foreground-Background (FB) Processing Schemes. Messages are stored at two different queues. Newly arrived messages are assigned to the first queue which is always given the higher priority for service. They are then given a fixed amount of service time and subsequently entered into the second queue. The latter is

served only when the first queue is empty. Differentiation is thus made between foreground and background service processes.

Multilevel Processor Sharing. The service discipline assigns a proper mode of scheduling rule to the message in the system according to the amount of service already given to the message. In this way a set of service discipline level is set up, so that the different levels are controlled between them by an FB algorithm.

We have noted the response time experienced by a message using the RR, FB and other related time-sharing schemes mentioned above, to be proportional to the required message processing time. To obtain this property, the various schemes utilized a feedback service procedure. In this way, a quantum of service is given to each message at a time. Such a procedure needs to be adopted when we have no prior knowledge concerning the message (or job, task) required processing time. The feedback scheme estimates this time through quantization.

However, if prior information is available concerning the required message service time, a much simpler nonfeedback structured scheme can be devised, incorporating this information, to yield the same message delay characteristics. This is many times the case in our system. Such priority service disciplines will be presented in this section.

We also note that the FCFS service discipline yields a message waiting-time which is independent of the message required processing time.

We can also consider a data processing system with a set of processors available to serve the messages. The queueing scheme

then involves multiple service channels. The resulting queueing characteristics are similar to those mentioned in Section I.4, except that the number of messages can be processed simultaneously, so that the multi-processor yields a higher service rate.

Another service discipline that could have been mentioned is the last-come first-served scheduling procedure. This discipline is noted to yield a response time vs required service time D(T) function which is identical to that obtained by a round-robin scheme. The schemes however yield different message delay variances.

We thus note, as will be observed again later, that to compare various priority service disciplines one needs to compute and compare also the variances associated with the message delays.

I.5.3 Service Disciplines for Messages in Different Priority Classes

Messages are many times classified into different priority classes. This classificationis affected by the message index of urgency and importance and by the message required response time.

Priority-1 (or class-1) messages have higher priority than priority-2 (or class-2) messages. In general, if there are P priority classes, we assign a higher priority to service class-k messages over class-j messages, whenever $k < j$. Messages belonging to the same priority class can be served according to sny pre-assigned priority procedure. In particular, we assume, unless stated otherwise, that a FCFS service discipline controls the service of messages belonging to the same priority class.

In considering the service of messages belonging to different priority classes by a single processing center, we can distinguish between the following disciplines.

Nonpreemptive Priority Discipline. Messages are ordered for service in accordance with their priority class. When the processor becomes available for service it accepts the message of highest priority. A FCFS ordering is used among messages of the same priority class. If, however, a higher priority message arrives at the system while a lower priority one is being processed, the latter is not preempted and its processing is allowed to be carried to completion.

Preemptive Resume Priority Discipline. The scheduling algorithm is as above except that if a newly arrived message belongs to a higher priority class than that presently in service, it is allowed to preempt the currently served message. The preempted message joins the queue, and when accepted for service its service resumes from the point it has been interrupted.

Preemptive Repeat Service Discipline. This scheduling procedure operates as the preemptive resume one except that the service of a message that has been previously preempted starts from the beginning. Thus, all the processing provided to a message is assumed lost if this message gets preempted by a higher priority message.

I.5.4 Analysis of a Priority Queueing System

Consider a system where messages are classified into 2 priority classes. Class 1 messages require high priority, while class 2 messages are of low priority.

Messages of class 1 arrive at random at the system according to the statistics of a Poisson process, with an arrival intensity of λ_1[mess./sec]. Class-2 messages arrive independently, also according

to a Poisson stream, with intensity λ_2 [mess./sec].

The system is assumed to provide a single server; i.e., a single processing facility. Messages of class 1 and 2 may require different (random) processing times. Thus, we set:

S_1 = average processing time required by a priority-1 message (I.5.4-1)

S_2 = average processing time required by a priority-2 message (I.5.4-2)

The corresponding second moments of the required message processing times are denoted as

$$\overline{S_1^2} = E(S_1^2) , \quad \overline{S_2^2} = E(S_2^2) \tag{I.5.4-3}$$

The average waiting time experienced by a randomly chosen message is denoted by W. Its average time delay (response-time) in the system is denoted by D. The average waiting time and time delay of a priority-1 message is W_1 and D_1, respectively. Similarly the average waiting time and time delay of a priority-2 message is W_2 and D_2, respectively.

We can write:

$$D = W + S ; \tag{I.5.4-4}$$

$$D_1 = W_1 + S_1 ; \tag{I.5.4-5}$$

$$D_2 = W_2 + S_2 , \tag{I.5.4-6}$$

where S is the average service time of a message chosen at random. Also, since a message chosen at random will belong to class L with probability λ_1/λ, where $\lambda = \lambda_1 + \lambda_2$, and to class 2 with probability $\lambda_2/\lambda = 1-\lambda_1/\lambda$, the following relationships hold

$$W = \frac{\lambda_1}{\lambda} W_1 + \frac{\lambda_2}{\lambda} W_2 \quad ; \qquad \text{(I.5.4-7)}$$

$$S = \frac{\lambda_1}{\lambda} S_1 + \frac{\lambda_2}{\lambda} S_2 \; ; \qquad \text{(I.5.4-8)}$$

$$D = \frac{\lambda_1}{\lambda} D_1 + \frac{\lambda_2}{\lambda} D_2 \; . \qquad \text{(I.5.4-9)}$$

The traffic intensities of the lower priority and higher priority schemes, ρ_2 and ρ_1, are given by

$$\rho_1 = \lambda_1 S_1 \; ; \qquad \text{(I.5.4-10)}$$

$$\rho_2 = \lambda_2 S_2 \quad . \qquad \text{(I.5.4-11)}$$

The traffic intensity ρ associated with the combined stream of arrivals is equal to

$$\rho = \rho_1 + \rho_2 = \lambda_1 S_1 + \lambda_2 S_2 \qquad \text{(I.5.4-12)}$$

We assume the processor to employ a non-preemptive priority service discipline.

If $\rho_1 \geq 1$, then high and low priority messages will experience arbitrarily long time delays as the system evolves in time. Thus,

$$W_1 = W_2 = \infty \quad \text{if } \rho_1 \geq 1 \; .$$

If however $\rho_1 < 1$, the higher priority message experiences a finite waiting time given by

$$W_1 = \frac{\lambda_1 \overline{S_1^2} + \lambda_2 \overline{S_2^2}}{2(1-\rho_1)} \quad , \quad \text{for} \quad \rho_1 = \lambda_1 S_1 < 1 \; . \qquad \text{(I.5.4-13)}$$

Also, we have

$$P(X_1=0) = P(W_1=0) = 1-\rho_1, \quad \text{for } \rho_1 < 1 \qquad \text{(I.5.4-14)}$$

where X_1 denotes the (queue-size) number of class 1 messages in the system. Thus, with probability $1-\rho_1$ there will be no higher priority messages in the system. The average delay of a priority message is given by (Pollaczek-Khintchine equation)

$$D_1 = W_1 + S_1 = \frac{\lambda_1\overline{S_1^2} + \lambda_2\overline{S_2^2}}{2(1-\rho_1)} + \overline{S_1}, \quad \text{for} \quad \rho_1 < 1 . \qquad \text{(I.5.4-15)}$$

The average number of priority messages in the system, denoted as $\overline{X_1}$, is equal (by Little's Theorem) to

$$\overline{X_1} = \lambda_1 W_1 + \rho_1 = \lambda_1 D_1$$

$$= \frac{\lambda_1^2\overline{S_1^2} + \lambda_1\lambda_2\overline{S_2^2}}{2(1-\rho_1)} + \rho_1 , \qquad \text{(I.5.4-16)}$$

where $\rho_1 = \lambda_1\overline{S_1} < 1$.

If

$$\rho = \rho_1 + \rho_2 \geq 1 ,$$

class-2 messages will experience arbitrarily long queueing delays, so that

$$D_2 = W_2 = \infty \quad \text{for } \rho \geq 1 \qquad \text{(I.5.4-17)}$$

For $\rho < 1$, the average waiting time for a lower priority message is given by

$$W_2 = \frac{\lambda_1\overline{S_1^2} + \lambda_2\overline{S_2^2}}{2(1-\rho_1)(1-\rho)} , \quad \text{for } \rho < 1 \qquad \text{(I.5.4-18)}$$

The response time of a lower priority message is thus equal to

$$D_2 = \frac{\lambda_1\overline{S_1^2} + \lambda_2\overline{S_2^2}}{2(1-\rho_1)(1-\rho)} + S_2 , \quad \text{for } \rho = \lambda_1 S_1 + \lambda_2 S_2 < 1 \qquad \text{(I.5.4-19)}$$

The average overall queue-size, i.e., average number of both class messages queueing in the system, is equal to

$$\overline{X} = \frac{\lambda[\lambda_1 \overline{S_1^2} + \lambda_2 \overline{S_2^2}]}{2(1-\rho)} + \rho, \quad \text{for } \rho < 1 \qquad \text{(I.5.4-20)}$$

The probabilty P(X=0) that the system will be totally empty is given by

$$P(X=0) = 1-\rho = 1 - \lambda_1 S_1 - \lambda_2 S_2 \quad , \qquad \text{(I.5.4-21)}$$

for $\rho < 1$. Therefore, the system index of utilization U, is expressed as

$$U = P(X > 0) = \rho = \lambda_1 S_1 + \lambda_2 S_2 \quad . \qquad \text{(I.5.4-22)}$$

Thus, the central processor is kept busy in serving (processing) both priority and regular messages a portion $U = \rho$ of the time, when $\rho < 1$. (For $\rho \geq 1$, clearly U=1).

The average waiting time W of a message chosen at random is given as

$$W = \frac{\lambda_1}{\lambda} W_1 + \frac{\lambda_2}{\lambda} W_2 = \frac{\lambda_1 \overline{S_1^2} + \lambda_2 \overline{S_2^2}}{2(1-\rho)} \quad \frac{1 - \frac{\lambda_1}{\lambda} \rho}{1 - \rho_1} \qquad \text{(I.4.5-23)}$$

for $\rho < 1$.

It can be noted that if $S_1 = S_2$, the waiting time W is identical to that obtained under a FCFS service discipline. The variance of the message waiting time, when considering a message chosen at random, is however lower when a FCFS scheme is used rather than a priority scheme. Of course, the priority scheme yields average waiting time values lower than W for higher priority jobs, while corresponding higher waiting time values are

experienced by lower priority tasks.

I.5.5 The Earliest Due Date Scheduling Discipline

It is necessary in a number of operation modes of the Space Shuttle avionics system to implement a dynamic priority queueing discipline. Subsequently, the computer and network resources are assigned to the users on a dynamic basis, based upon the current state of the network, the current queue sizes and experienced job delays and the current requirements for service. The job currently in the queue is chosen to be processed by the computer system in accordance with its spent waiting time in the system, priority, required response time, required service duration and the similar characteristics of the other jobs presently queued for service.

A general model of such a dynamic priority service discipline, which is particular important for the proper operation of the Space Shuttle data network in high traffic and critical phases, is described in the following. It is described as an Earliest Due Date (EDD) service discipline.

Jobs (or task, or messages) arriving at the processor are classified into k classes. A class i job is associated with an urgency number u_i, i = 1,2,...,k. Let

$$u_1 \leq u_2 \leq \ldots \leq u_k \qquad \text{(I.5.5-1)}$$

The lower the urgency number, the more urgent is the required service. If a class i job arrives at the system at time t_i, he is assigned a real number

$$d_i = t_i + u_i \, . \qquad \text{(I.5.5-2)}$$

This number d_i^{-1} can be regarded as a dynamic priority number. Under an associated head of the line discipline, the processor admits into service the job with the minimum value of $\{d_i=t_i+u_i\}$. Ties can be broken by choosing the job with the minimum urgency number. If preemption of a job (from service) is allowed, the scheduling procedure is modified so that the system is continuously monitored, and the job that is being processed has the minimum value of $\{t_i+u_i\}$ out of all jobs in the system.

In comparing this dynamic priority scheduling rule with the static priority discipline presented in Sections I.5.3-I.5.4, we note the following. In applying the dynamic queueing rule, the u_i's serve the purpose of distinguishing between static priority classes. Thus, a class 1 job is of highest static priority and a class k job has the lowest static priority. But in addition a job that has been waiting for service as reflected by its arrival time t_i gains in priority dynamically over time.

For our purposes, it is generally convenient to let the u_i's correspond to the interval until the due date is reached. Thus, a class i job arriving at time t_i has a <u>due date</u> t_i+u_i desired for receiving service. Subsequently, we can choose u_i to reflect the desired response time and priority of class i jobs, in relation to the other jobs.

As such, this priority scheme is noted to realize scheduling by the earliest due date (EDD) rule in the processing queueing system.

As special cases, if we set $u_i=0$ for each i, this service discipline becomes a FCFS scheduling rule, while if $u_2-u_1 = +\infty$

we have a static priority service procedure where class 1 jobs are always processed ahead of class 2 jobs present at the same time. Thus, by changing the difference in urgency numbers from 0 to + , the discipline evolves from a FCFS one to a static priority one.

We indicate now a few performance characteristics associated with the EDD discipline. Assume k=2, so that we have only 2 classes of jobs. Class 1 jobs are higher priority jobs with an urgency number u_1-u_h. Class-2 jobs have lower priority and an urgency number $u_2 = u_\ell$. We let $W_h(t)$ denote the waiting time (in the queue, prior to initiation of service) of a higher-priority (class 1) job arriving (virtually) at time t. Similarly, we let $W_\ell(t)$ be the waiting time of a class-2 lower priority job at time t. We can also consider a non-preemptive or preemptive-resume service discipline. The waiting time at t of a higher priority job under preemptive-resume and non-preemptive discipline is denoted as $W_{h,p}(t)$ and $W_{h,n}(t)$, respectively. The waiting time $W_\ell(t)$ of the lower priority job is clearly independent of whether a preemptive or non-preemptive procedure is employed.

We find that for $t \leq u_\ell$,

$$W_\ell(t-u_\ell)-u_\ell \leq W_{h,p}(t-u_h)-u_h \leq W_{h,n}(t-u_h)-u_h \ . \qquad \text{(I.5.5-3)}$$

To demonstrate further these inequalities we define the lateness of a lower priority job $L_t(t)$ and higher priority job $L_{h,n}(t)$, $L_{h,p}(t)$, at t, by

$$L_\ell(t) = W_\ell(t) - u_\ell \qquad \text{(I.5.5-4a)}$$

$$L_{h,n}(t) = W_{h,n}(t) - u_h \quad , \qquad \text{(I.5.5-4b)}$$

$$L_{h,p}(t) = W_{h,p}(t) - u_p \quad . \qquad \text{(I.5.5-4c)}$$

Thus the lateness L(t) of a job at time t describes the difference between the job waiting time in the queue and its urgency value. If the urgency value corresponds to a desired expected job waiting time, then the lateness variable describes the deviation of the job waiting time from its desired expected value. We then have, for $t \geq u_\ell$,

$$L_\ell(t-u_\ell) \leq L_{h,p}(t-u_h) \leq L_{h,n}(t-u_h) \quad . \qquad \text{(I.5.5-5)}$$

Thus, a lower priority job arriving at time $t-u_\ell$ will be served no later than a higher priority job arriving at time $t-u_h \geq t-u_\ell$. If the class-2 (lower priority) job waits at least $u_\ell-u_h$ units of time, his due date becomes the same as that of a class-h job arriving $u_\ell-u_h$ units later, and the jobs are then of equivalent priority. Subsequently, equality occurs above and the above mentioned class-1 and class-2 jobs experience the same lateness values.

Thus, note that lower priority (class 2) jobs increase their dynamic queueing priority index after waiting in the queue $u - u_h$ units of time so that at this time they attaindynamic priority equivalent to that of higher priority (class 1) jobs.

To indicate explicit analytical results, we assume class-h and class-ℓ jobs to arrive according to Poisson streams with intensities λ_h [jobs/sec] and λ_ℓ [jobs/sec], respectively. Also assume required processing times of S_h [sec/job] and S_ℓ [sec/job]

for high priority and low priority jobs, respectively. The related moments of the required processing times are denoted as $E(S_h)$, $E(S_\ell)$, $E(S_h^2)$ and $E(S_\ell^2)$. The traffic intensities are

$$\rho_h = \lambda_h E(S_h),\ \rho_\ell = \lambda_\ell E(S_\ell) . \qquad \text{(I.5.5-6)}$$

If

$$\rho_h + \rho_\ell < 1 , \qquad \text{(I.5.5-7)}$$

as we assume henceforth, the system will enter steady-state where finite job waiting-times are experienced.

Let

$$u = u_\ell - u_h . \qquad \text{(I.5.5-8)}$$

We denote by $B_h(W)$ the busy-period duration spent in servicing only newly arriving class-h jobs, starting with an initial service load of W sec. Then the (steady-state) mean waiting times of higher priority jobs under a non-preeemptive rule, $E(W_{h,n})$, and of low priority jobs, $E(W_\ell)$, are given as follows.

$$E(W_{h,n}) = E(W) - \rho_\ell \int_0^u P\{B_h(W) > y\}dy , \qquad \text{(I.5.5-9)}$$

$$E(W_\ell) = E(W) + \rho_h \int_0^u P\{B_h(W) > y\}dy , \qquad \text{(I.5.5-10)}$$

where $E(W)$ is the mean waiting-time of an arbitrary job in the combined-traffic queueing system, given by the Pollaczek-Khintchine formula as

$$E(W) = \frac{\lambda_h E(S_h^2) + \lambda_\ell E(S_\ell^2)}{2(1-\rho_\ell-\rho_h)} \qquad \text{(I.5.5-11)}$$

The distribution of the busy-period $B_h(W)$ can be calculated by considering the associated (high priority) M/G/1 queueing system. In particular, the mean busy-period duration is equal to

$$\int_0^\infty P\{B_h(W) > y\}dy = E[B_h(W)] = \frac{E(W)}{1-\rho_h} \qquad \text{(I.5.5-12)}$$

We can thus write for each $u \geq 0$,

$$\int_0^\infty P\{B_h(W) > y\}dy = g(u)\frac{E(W)}{1-\rho_h} \qquad \text{(I.5.5-13)}$$

The function $g(u)$ is defined by the above equation, and is clearly a continuous monotone increasing function of u assuming values in $[0,1]$:

$$0 \leq g(u) \leq 1,\ g(0) = 0,\ \ g(\infty) = 1. \qquad \text{(I.5.5-14)}$$

Substituting in (I.5.5-9)-(I.5.5-10), we obtain

$$E(W_{h,n}) = E(W)\left[1-g(u)\frac{\rho_\ell}{1-\rho_h}\right] \qquad \text{(I.5.5-15)}$$

$$E(W_\ell) = E(W)\left[1 + g(u)\frac{\rho_h}{1-\rho_h}\right] \qquad \text{(I.5.5-16)}$$

In particular, we observe that

$$E(W_\ell) - E(W_{h,n}) = E(W)g(u)\frac{\rho_\ell+\rho_h}{1-\rho_h}\ , \qquad \text{(I.5.5-17)}$$

yielding the difference in average waiting times between lower priority and higher priority jobs, using an EDD service discipline with $u = u_\ell - u_h$.

Consider now the two extreme special cases. If $u = u_\ell - u_h = 0$, we have a FCFS service discipline, and then $g(u) = g(0) = 0$ so that

$$E(W_{h,n}) = E(W_h) = E(W) . \qquad \text{(I.5.5-18)}$$

Thus, no priority classes are being distinguished, and the average waiting time of any job is given by (I.5.5-11).

If $u = u_\ell - u_h \to \infty$, then $g(u) \to g(\infty) = 1$, and we obtain

$$E(W_{h,n}) = \frac{\lambda_h E(S_h^2) + \lambda_\ell E(S_\ell^2)}{2(1-\rho_h)} , \qquad \text{(I.5.5-19)}$$

$$E(W_\ell) = \frac{\lambda_h E(S_h^2) + \lambda_\ell E(S_\ell^2)}{2(1-\rho_h)(1-\rho_h-\rho_\ell)} . \qquad \text{(I.5.5-20)}$$

These are the same equations as noted in a previous section for the average waiting-times of high and low priority jobs in a system with two stationary priority classes.

It is obvious by (I.5.5-17) that by choosing the urgency difference number $u = u_\ell - u_h$, we can obtain a desired difference $E(W_\ell) - E(W_{h,n})$ between the average waiting-times of low and high priority jobs. This difference is 0 when $u=0$, $g(0)=0$, and FCFS procedure is used. The difference attains its maximal value at $u=\infty$, $g(\infty)=1$, when stationary priorities are used.

For example, if $\rho_h=0.5$, $\rho_\ell=0.4$, $\rho = \rho_h+\rho_\ell = 0.9$, $E(W)$ is relatively high and when 2 stationary priorities are used, $u=\infty$, $g(\infty)=1$, we have

$$E(W_\ell) - E(W_{h,n}) = 1.8E(W) .$$

This difference can be high for certain applications. By using an EDD scheduling rule we can choose $0 \leq g(u) \leq 1$ to lower the latter difference. For example, we can set $u = u_\ell - u_h$ to yield $g(u) = 0.2$, and then

$$E(W_\ell) - E(W_{h,n}) = 1.8E(W)0.2 = 0.36E(W) ,$$

which can be acceptable.

It should be noted that although class-1 jobs experience shorter waiting times than class-2 jobs, the same is not true regarding the corresponding lateness values. In fact, the lateness variable L_ℓ of a low priority job is stochastically smaller than the lateness variable L_h (representing $L_{h,p}$ or $L_{h,n}$) of a high priority job. Thus,

$$P\{L_\ell \geq x\} \leq P\{L_h \geq x\} . \qquad \text{(I.5.5-21)}$$

Hence,

$$E(L_\ell) \leq E(L_h) = E(W_h) - u_h \leq E(W) - u_h . \qquad \text{(I.5.5-22)}$$

This property that jobs from the class with the earliest due date have the maximum mean lateness, though having the shortest waiting times, is desirable from the system's point of view in meeting the most urgent needs of the jobs.

In designing an earliest due date rule we can properly optimize the choice of the urgency (due date) parameters $\{u_i\}$, as illustrated by the following. Assume 2 priority classes, and non-preemptive EDD service disciplines. Let $c_1 > 0$, $c_2 > 0$ represent costs per unit of waiting time for class 1 and class 2 jobs, respectively. An overall cost value C is chosen then as

$$C = c_1E(W_1) + c_2E(W_2) . \qquad \text{(I.5.5-23)}$$

We wish to choose u_1, u_2 to minimize C. By the above expressions, we conclude that we need to minimize

$$(c_2\rho_1 - c_1\rho_2) \int_0^{u_2-u_1} P\{B_1(W) > y\}dy \quad .$$

Subsequently, we conclude that C is minimized over all dynamic priority queue-disciplines if we choose:

$$u_2-u_1 = 0 \text{ (FCFS, if } c_2/c_1 > \rho_2/\rho_1 \text{ ;}$$

$$u_2-u_1 = \infty \text{ (static priority), if } c_2/c_1 < \rho_2/\rho_1; \qquad \text{(I.5.5-24)}$$

and dynamic priority discipline, if $c_2/c_1 = \rho_2/\rho_1$.

In particular, if we set

$$c_1 = a_1 \frac{\lambda_1}{\lambda_1+\lambda_2} \quad , \quad c_2 = a_2 \frac{\lambda_2}{\lambda_1+\lambda_2} \quad , \; a_1 > 0, \; a_2 > 0, \qquad \text{(I.5.5-25)}$$

then, if

$$E(S_1)/a_1 \leq E(S_2)/a_2 \quad , \qquad \text{(I.5.5-26)}$$

the optimal policy is to set $u_2-u_1 = \infty$ and use a static priority discipline. Thus, we then attach always higher priority to jobs whose weighted (by a_i) requried processing times are shorter.

I.6 THE COMPUTER SYSTEM: QUEUEING MODELS AND PERFORMANCE ANALYSIS

I.6.1 Operating Systems

We consider a computer system, such as that associated with the general purpose computer (GPC) of the Space Shuttle avionics system.

The term "process" is used to denote a program in execution.

The computer system can be defined in terms of the various supervisory and control functions it provides for the processes created by its users:

a. Creating and removing processes.

b. Controlling the progress of processes.

c. Responsing to irregular conditions that may occur during the execution of the process, such as: interrupts, arithmetic or machine or addressing errors, protection violations.

d. Allocating hardware resources among processes.

e. Providing access to software resources.

f. Providing protection, access control and information security.

g. Providing interprocess communication and synchronization.

The computer system software that assists the hardware in implementing these functions is known as the operating system.

To become an efficient processing system, a computer system will generally incorporate the following characteristics:

a. Concurrency - parallel processing.

b. Automatic resource allocation.

c. Sharing of resources by more than one process.

d. Multiplexing of information over an access channel, and providing remote conversational access to system resources or processes.

e. Asynchronous operation.

f. Long term storage of information; e.g., in the form of a file system.

These characteristics involve the management of the computer memory and processes. Algorithms used to be efficiently designed for:

a. Managing, controlling and schedulgin processes;

b. Managing and controlling main and auxiliary memory devices;

c. Managing and controlling the flows of information among the various devices in a computer system.

The two important major sets of resources for the computer system are processor resources and memory resources. A processor is any device which handles information or carries out the steps of a process, such as: central processing unit, arithmetic processor, I/O (input/outout) processor or an access channel A memory is a device which is used for storage of information. The capacity of a memory device is the number of words (or bytes, or bits) of information that it can store. The access time of a memory device is the average time duration between the receipt and completion of a "memory-fetch" request, when queueing delays are neglected. A memory device is random access if the access time of each storage site is the same; examples: semiconductor

and core memories. A memory device is positionally addressed if the access time of a word depends on its positon; examples: disks, drums and tapes.

Informationis generally stored in a computer system in a two level storage system: main memory and auxiliary memory. Information residing in main memory is usually random access and requires very short access time, so that it can be immediately accessible for processing. Otherwise, it resides in auxiliary memory which is usually positionally addressed and requires relatively longer access times.

For the GPC on the Space Shuttle, the main memory is composed of pluggable, random-access, non-volatile, destructive-read-out ferrite core modules with a monolithic option. The access time for this memory system is:

$$\text{access time} = 0.375\ \mu\text{sec} .$$

The capacity of the memory is:

$$\text{capacity} = 1310720 \text{ bits} = 40960 \text{ words}$$

where the word length is:

$$\text{data word length} = 16/32 \text{ bits (fixed point)} = 32/64 \text{ bits (floating point)} ;$$

$$\text{instruction word length} = 16/32 \text{ bits} .$$

Also, for this GPC we have:

$$\text{number of instructions in repetoire} = 154;$$

$$\text{computing speed } 480 \times 10^3 \frac{\text{operations}}{\text{sec}} \text{ (fixed point)} ; = 325 \times 10^3 \frac{\text{operations}}{\text{sec}} \text{ (floating point)}.$$

On the Space Shuttle orbiter, two high capacity tape units are also used as mass memory. The storage capacity of each is 134 megabits of data. They are used to store permanent on-board off-line information. They thus supplement the on-line random-access internal memories of the Space Shuttle computers.

Process coordination characteristics are important in designing a computer system and assessing its performance. In a multiprogramming system, both process interruption at arbitrary times and peripheral activity of arbitrary speed are carried simultaneously. It is thus necessary to guarantee that the computation performed when cooperating processes are involved is independent of the relative speeds of the different tasks. Computation then is required to be determinate. In addition, in considering process coordination and control problems, one should study the following problems: deadlocks; mutual exclusions between tasks; and synchronization objectives, needed for example to ensure the timing of the proper start of a certain procedure in correspondence with the occurrence of a certain event.

I.6.2 Memory Management

A memory management algorithm is composed mainly of the following policies:

a. The fetch policy determining when a block is transferred from auxiliary to main memory.

b. The placement policy determining the unallocated space of main memory into which an incoming block is to be palced.

c. The replacement policy determining which blocks are to be removed from the main memory.

The structure of a two level memory system is shown in Fig. I.6-1. The above mentioned policies are implemented by move commands which control the moving of blocks between main and auxiliary memories.

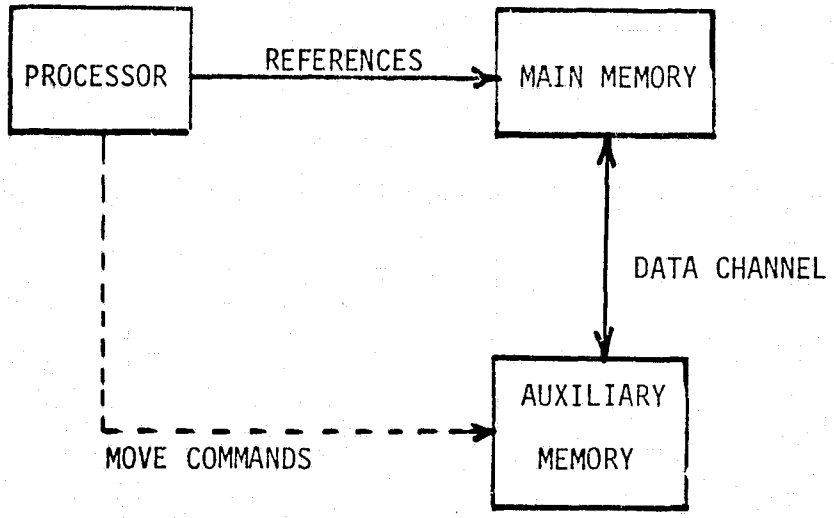

Figure I.6-1.

To analyze the memory management procedure used in the GPCs of the Space Shuttle avionics system, it is particularly useful to use a virtual memory technique.

A virtual memory can be regarded as the main memory of a simulated (or virtual) computer. A virtual memory system is described in terms of two spaces, N and M, and a mapping f. The address space N of a task is the set of addresses that can be generated by a processor as it executes the task. Tasks can share the same address space. In multiprogramming systems, several address spaces are utilized. The memory space M of the system represents the set of locations in the physical main memory.

The address map f provides for the transformation

$$f: \quad N \rightarrow M \cup \{\phi\}$$

from space N to space M or to a set $\{\phi\}$. Sets $\{\phi\}$ indicates that the desired word is presently not in the memory space. Thus, if x is an address in N, then if

$$f(x) = y \varepsilon M,$$

the desired address is stored in main memory at location y at that time. If, however, $f(x) \notin M$, or $f(x) \varepsilon \{\phi\}$, the desired address is not in main memory, and a fault condition results. Move commands are then initiated and the table describing f is adjusted. This is illustrated in Fig. I.6-2.

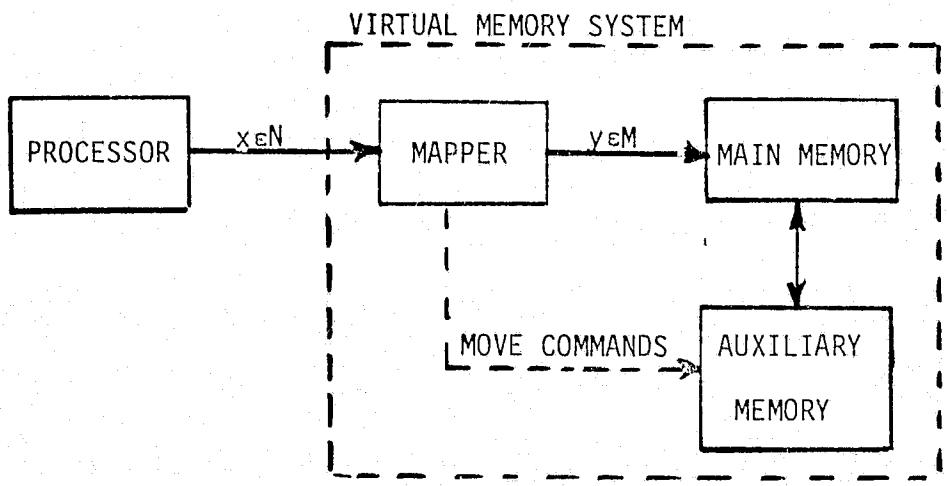

Figure I.6-2.

In analyzing the performance of auxiliary memory systems, one considers mainly the underlying queueing problems. This is the case due to the relatively long access times involved. Subsequently, such memory units can become congestion centers within the computer system. The models and analysis techniques invovled are similar to those presented in the sections on queueuing models and analysis. The index of performance usually used in choosing the related optimal scheduling algorithm is that of maximizing the throughput of the memory subsystem.

In our applications the problems mentioned above concerning data transfers between main memory and auxiliary memory arise also in connection with data flows between main memory and I/O devices. In this connection, one needs also to consider the associated buffer problems. Shared (pooled) buffers are much more efficient than individual dedicated buffers. The proper related performance criteria here are buffer occupancy and overflow probability. We note that under multiprogramming the main memory can be regarded as a shared buffer.

In studying main memory management the objective is usually related to maximum execution speed of programs.

I.6.3 On Computer Scheduling Procedures

In modeling and studying processor scheduling procedures we can distinguish between deterministic scheduling rules and probabilistic scheduling models.

In considering deterministic scheduling disciplines we assume that we are given a (partially ordered) set of tasks whose execution (required processing) times $\{S_i\}$ are known. We also assume that there are m (identical) processors available to execute these tasks.

Two performance measures are then considered: the time until the last task is completed and the average turnaround (flow) time.

The first measure is related to the system utilization factor U. Thus, if a given schedule finishes in time T, the utilization factor the processors by the schedule is

$$U = \frac{\sum_i S_i}{mT} .$$

Hence, minimizing T is equivalent to maximizing T.

The second measure is of interest to the users of the system. It many times also yields the minimization of the average number of incomplete tasks.

The deterministic models assume that the processing times of all tasks are known in advance and that all tasks are available for execution at once. It is more realistic in our applications to assume that the processing times required by the various tasks are random, governed by certain probability distributions. Also, we then assume that tasks arrive at the processing system at random times. We then need to specify the joint statistics of the task inter-arrival times.

Using these probabilistic characterizations of the tasks arrival streams and required execution times, the processing system is modeled as a queueing system. The associated service discipline then represents the task scheduling rule.

Queueing models have been presented, discussed and analyzed in Section I.4. In particular, time-sharing queueing models have been considered. Priority service disciplines have been classified, discussed and analyzed in Section I.5.

In assigning priorities to tasks, we associate an index of preference or urgency to the processing of a task relative to other tasks. As noted in Section I.5, these priority or urgency indices can be assigned on a static basis or dynamically in accordance with the state of the system and the task desired response time or actual current lateness.

Systems can use "time slicing" to limit the length of

processing time that can be given to a task at one time. Tasks which use the processor at one time more than a certain quantum duration, are interrupted and asked to release the processor. They are then reassigned for service in accordance with the system service discipline.

In particular, we have considered the following service disciplines:

1. First-come first-served (FSFS).
2. Round-robin foreground-background and multilevel service procedures.
3. Service disciplines for tasks classified into fixed priority classes.
4. Earliest due date dynamic priority queueuing disciplines.

In addition, one can incorporate service disciplines that assign dynamic priorities in accordance with task processing times; giving, for example, priority to shorter tasks over longer ones. Or, as we already noted, giving service to tasks that have currently received the least amount of overall service.

We present and study in the next sections certain queueing models that can be used for performance prediction in our data processing system.

I.6.4 A Markovian Queueing Model: Finite Buffer Facility

We present in this and the next 2 sections a simple Markovian queueing model. We also present its performance characteristics. It is noted that this model can be used for a first-order performance prediction. It allows the incorporation of arrival and service rates that depend upon the state of the system. Subsequently,

one can use it to analyze a processing system with a finite buffer size, multiple processors and an arrival task stream that is generated by a finite source of users (or terminals).

We assume tasks (or jobs, or messages, or customers) to be exponentially distributed with mean required task processing time being $1/\mu$ [sec/task]. Tasks arrive at the processor according to a Poisson stream.

Assume a single processing unit with a finite buffer facility, with storage space for at most L tasks. Tasks arriving when the buffer is full are assumed to be rejected. Let p_n denote the probability that n tasks are in the system (at steady state), queueing or being processed. Then, we have:

$$p_n = \frac{\rho^n(1-\rho)}{1-\rho^{L+1}} \quad , \quad n = 0,1,\ldots,L \ . \tag{I.6.4-1}$$

where $\rho = x/\mu$. In particular, the system utilization index U describing the probability that the processor is busy is given by

$$U = 1-p_0 = 1 - \frac{1-\rho}{1-\rho^{L+1}} = \frac{(1-\rho^L)}{1-\rho^{L+1}} \ . \tag{I.6.4-2}$$

The average queue size $\bar{X}$ is given as

$$\bar{X} = \sum_{n=0}^{L} np_n = \frac{1-\rho}{1-\rho^{L+1}} \sum_{n=0}^{L} n\rho^n \ . \tag{I.6.4-3}$$

The probability p_R that a task is rejected from the system, not accepted for service due to a full buffer, is given by

$$p_R = p_L = (1-\rho) \frac{\rho^L}{1-\rho^{L+1}} \quad , \ 0 < \rho < \infty. \tag{I.6.4-4}$$

Clearly, $p_R \to 0$ as $L \to \infty$, while $p_R = \rho$ for $L = 1$. The character of the rejection probability (or overflow probability) curve, as a

function of the buffer size L-1, is shown in Fig. I.6-3.

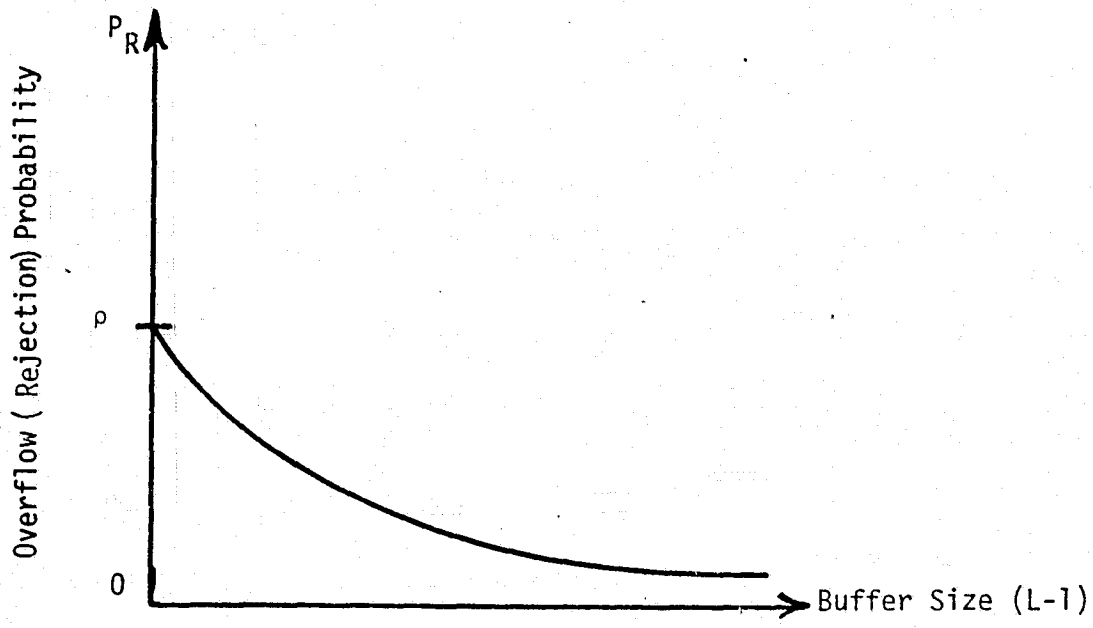

Figure I.6-3.

The average waiting time W of a task, provided this task is accepted into the system (i.e., the buffer is not full) is expressed as

$$W = \sum_{n=0}^{L-1} n\mu^{-1} \frac{p_n}{1-p_L} \quad . \qquad \text{(I.6.4-5)}$$

We thus note that a too small buffer size (L small) can imply a very high overflow probability, or rejection probability, as illustrated by eq. (I.6.4-4) and Fig. I.6-3. However, increasing the buffer size beyond a large enough value L* would not significantly improve the overflow probability.

If a rejection probabilty no higher than p is desired,

$$p_R \leq p \ , \qquad \text{(I.6.4-6)}$$

then by (I.6.4-4) we should set the buffer size L-1 according to the formula

$$L = \frac{\log\left[\frac{p}{1-\rho(1-p)}\right]}{\log \rho} \qquad \text{(I.6.4-7)}$$

Note that L-1 is the capacity of the buffer facility measured in number of messages. The average capacity in bits, L_B, is obtained as follows. Assume the (average) processing rate of the service facility to be

$$\text{processor rate} = C \text{ [bits/sec]} \ . \qquad \text{(I.6.4-8)}$$

Then since the average task required processing time is

$$\text{average task required processing time} = \mu^{-1} \text{ [sec/task]} \ , \qquad \text{(I.6.4-9)}$$

we conclude that

$$\text{average task length in bits} = C\mu^{-1} \text{ [bits/task]} \ . \qquad \text{(I.6.4-10)}$$

Therefore, the capacity of the storage facility in bits is

$$L_B = LC\mu^{-1} \text{ [bits]} \ . \qquad \text{(I.6.4-11)}$$

Also note that as the storage capacity L is decreased the average queue size $\overline{X}$ and the average waiting time $\overline{W}$ of an accepted task both decrease, since accepted tasks have to content for service with less other accepted tasks. The overflow probability then of course decreases as well.

I.6.5 A Finite Task Source Queueing Model

It is necessary for our applications to be able to model, at certain operational modes of the data processing system, part of the incoming task stream as composed of a finite set of sources.

For that purpose, assume that the processor experiences an arrival stream that originates from a set of N task sources (or terminals). Between the completion of its previous task and the submission of a new task to the processor a certain random delay time is generally noted. This time delay is called the "think time" source.

We assume here that the think time of each terminal (task source), of the N terminals, is exponentially distributed with mean λ^{-1}; i.e.,

$$\text{Average source think time} = \lambda^{-1} \text{ [sec]} \qquad \text{(I.6.5-1)}$$

The system model is shown in Figure I.6-4. We note that if there are currently n tasks in the system, $n \leq N$, only N-n new tasks can presently arrive (according to a Poisson stream with intensity $(N-n)\lambda$).

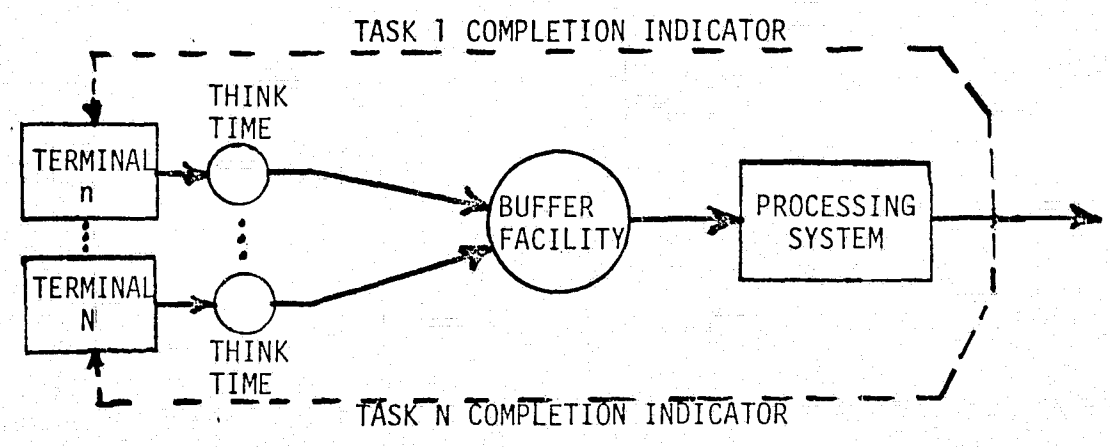

Figure I.6.4.

We assume, as in the previous section, that task processing times are exponentially distributed with mean μ^{-1}; i.e.,

$$\text{Average required task processing time} = \mu^{-1} \text{ [sec/task]} \qquad \text{(I.6.5-2)}$$

Let

$$\rho = \lambda/\mu \qquad \text{(I.6.5-3)}$$

be the traffic intensity parameter. We also set

$$P_n = P\{n \text{ tasks in the system}\} , \qquad \text{(I.6.5-4)}$$

at steady-state, considering both the task in service and the tasks waiting in the queueing facility. Then, we obtain

$$P_0 = \left\{ \sum_{i=0}^{N} \frac{N!}{(N-i)!} \rho^i \right\}^{-1} , \qquad \text{(I.6.5-5a)}$$

$$P_n = P_0 \rho^n \frac{N!}{(N-n)!} , \; n = 0,1,\ldots,N . \qquad \text{(I.6.5-5b)}$$

The system utilization index U is computed as

$$U = P\{\text{processor busy}\} = 1-P_0$$

$$= 1 - \left\{ \sum_{i=0}^{N} \frac{N!}{(N-i)!} \rho^i \right\}^{-1} . \qquad \text{(I.6.5-6)}$$

The task average waiting time W is equal to

$$W = \mu^{-1}\overline{X} , \qquad \text{(I.6.5-7)}$$

where $\overline{X}$ is the average queue size,

$$\overline{X} = \sum_{n=1}^{N} nP_n . \qquad \text{(I.6.5-8)}$$

If we also assume a finite storage capacity so that

$$\text{Number of task in system} \leq L < N \quad , \tag{I.6.5-9}$$

counting both tasks in the queueing facility and the task in service, we obtain the queue-size probabilities to be given as follows.

$$P_0 = \left\{ \sum_{i=0}^{L-1} \frac{N!}{(N-i)!} \rho^i \right\}^{-1} \quad , \tag{I.6.5-10a}$$

$$P_n = P_0 \rho^n \frac{N!}{(N-n)!} \quad , \quad n = 0,1,\ldots,L \; . \tag{I.6.5-10b}$$

The average queue size $\overline{X}$ is computed using Eqs. (I.6.5-8) and (I.6.5-10). The average waiting time of an accepted message is computed by

$$W = \sum_{n=0}^{L-1} n\mu^{-1} \frac{P_n}{1-P_L} \quad . \tag{I.6.5-11}$$

The probability P_R that a task will be rejected due to a full buffer (or the buffer will overflow) is equal to

$$P_R = P_L = \left\{ \sum_{i=0}^{L-1} \frac{N!}{(N-i)!} \rho^i \right\}^{-1} \rho^L \frac{N!}{(N-L)!} \tag{I.6.5-12}$$

Using these expressions, one can properly design the data processing system. In particular, if a maximum overflow probability P_R is specified,one can compute by (5.12) the desired buffer size. The latter is equal to L-1 messages or $(L-1)C\mu^{-1}$ bits, where C is the processing rate (in bits/sec) of the service system. The system utilization index U is now given by

$$U = P\{\text{processor is occupied}\}$$

$$= 1 - P_0 = 1 - \left\{ \sum_{i=0}^{L-1} \frac{N!}{(N-i)!} \rho^i \right\}^{-1}, \qquad \text{(I.6.5-13)}$$

where $\rho = \lambda/\mu$.

I.6.6 A Multi-Processor Queueing Model

In certain operational modes of the data processing system, we need to consider the situation where a task can be processed by any one of a set of processors. We thus present a Markovian queueing model to describe the queueing system performance characteristics in this situation.

Assume the system to contain m identical processors (service units). Arriving tasks (or requests for processing) are stored in a queue if all m processors are busy. As soon as a processor becomes free, it accepts into service the task of the head of the queue. The system is illustrated in Fig. I.6-5.

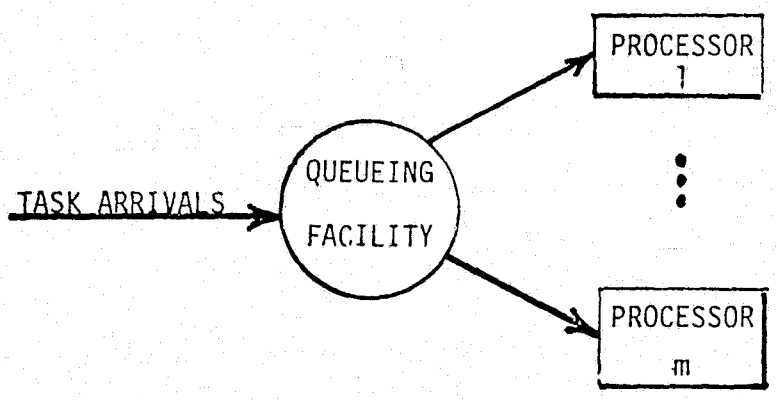

Figure I.6-5.

Assume tasks (or messages) to arrive at the system according to a Poisson stream with intensity λ [tasks/sec]. We also take the average requried processing time for a task to be μ^{-1} [sec/tasks].

The required task processing time is assumed to be exponentially distributed.

We then obtain the queue-size probabilities $\{P_n\}$, where

$$P_n = P\{\text{n tasks in the system, queueing or being served}\}, \quad \text{(I.6.6-1)}$$

to be given by the following expressions

$$P_0 = \left\{ \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} \right\}^{-1} ; \quad \text{(I.6.6-2)}$$

$$P_n = \begin{cases} \frac{(m\rho)^n}{n!} P_0 , & n < m \\ \frac{m^m}{m!} \rho^n P_0 , & n \geq m \end{cases} \quad \text{(I.6.6-3)}$$

where

$$\rho = \lambda/m\mu \quad \text{(I.6.6-4)}$$

and $\rho < 1$.

The system index of utilization U giving the fraction of time that the system is occupied (so that at least one processor is busy) is given by

$$U = P\{\text{at least one processor is busy}\}$$

$$= 1 - P_0 = 1 - \left\{ \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} \right\}^{-1} , \quad \text{(I.6.6-5)}$$

with $\rho = \lambda/m\mu < 1$.

The fraction of time that all processors are busy, denoted as U_m, is equal to

$$U_m = P\{\text{all } m \text{ processors are occupied}\}$$

$$= \sum_{n=m}^{\infty} P_n = 1 - \sum_{n=0}^{m-1} P_n$$

$$= 1 - P_0 \sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!}$$

$$= 1 - \left\{ \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} \right\}^{-1} \sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} \quad . \qquad (I.6.6\text{-}6)$$

The average task waiting-time W is computed as

$$W = (m\mu)^{-1} \sum_{n=m}^{\infty} nP_n \quad . \qquad (I.6.6\text{-}7)$$

For given message and traffic statistical parameters (λ and μ), we note that by increasing the number of parallel processors m we decrease the task waiting time (and subsequently reduce its response time). However, at the same time we obtain a reduced value for the index of utilization U_m (or U). In designing the system, one then chooses the number of parallel processors m properly, using Eqs. (I.6.6-2)-(I.6.6-7) so that a high enough index of utilization is achieved while an acceptable task response time is guaranteed.

As another useful model for the Space Shuttle processing subsystem, assume now that we have m parallel processors as above, but that the arrival stream is generated by a finite set of sources. As in the previous section, we set the number of task sources to be equal to N. The terminal thinking time is taken to be an exponentially distributed random duration with mean λ^{-1} [sec]. Required task processing times are exponentially distributed with

means μ^{-1} [sec/task].

Then, the probabilities $\{p_n\}$ of the number of tasks in the system, being processing or waiting in the queueing facility is computed to be given by the following formulas. For

$$\rho = \lambda/m\mu < 1 ,$$

we have

$$P_0 = \left\{ \sum_{n=m}^{\infty} \frac{m^m}{m!} \rho^n \frac{N!}{(N-n)!} + \sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} \frac{N!}{(N-n)!} \right\}^{-1} , \quad \text{(I.6.6-8)}$$

$$P_n = \begin{cases} P_0 \frac{(m\rho)^n}{n!} \frac{N!}{(N-n)!} , & \text{if } n < m \\ P_0 \frac{m^m}{m!} \rho^n \frac{N!}{(N-n)!} , & \text{if } n \geq m \end{cases} \quad \text{(I.6.6-9)}$$

The index of utilization U_m is given as

$$U_m = P\{\text{all m processors are busy}\}$$

$$= 1 - \sum_{n=0}^{m-1} p_n . \quad \text{(I.6.6-10)}$$

The average task waiting time W is computed using Eq. (I.6.6-7).

We again note that the latter equations should be used to design the system such that the proper acceptable system utilization and message response times are deduced.

I.6.7 Queueing Models Involving Input/Output and CPU Interactions

In studying the performance of the Space Shuttle computer system, it is of particular importance to incorporate the interactions between the input/output and CPU queues. Proper queueing models for

describing these interactions, and their performance characteristics and formulas, will be presented in this section.

A basic simple model is that of a cyclic queue involving single CPU and I/O processors, shown in Fig. I.6-6.

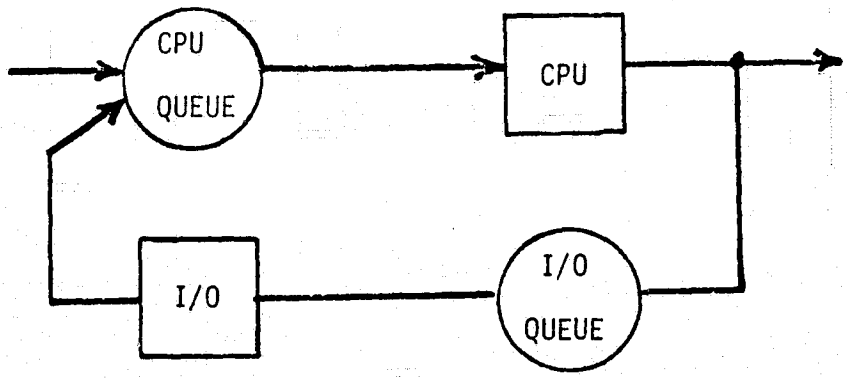

Figure I.6-6

Tasks enter the system by joining the CPU queue, but only at the instants when tasks depart from the system. In that way the number of tasks in the system is kept constant at N. After receiving service by the CPU a task leaves the system with probability α, and then a new task immediately enters the system. With probability $1-\alpha$ the task processed by the CPU enters the I/O queue. There, tasks are served on a first-come first-served basis. Upon departure from the I/O processor, a task joins immediately the CPU queue.

We assume that each task is assigned a processing time by the CPU and I/O which are independent and exponentially distributed, with means μ_C^{-1} and μ_I^{-1}, respectively. Thus,

$$\text{Avg. CPU service time} = \mu_C^{-1} \text{ [sec]} ; \qquad \text{(I.6.7-1)}$$

$$\text{Avg. I/O service time} = \mu_I^{-1} \text{ [sec]} . \qquad \text{(I.6.7-2)}$$

Since a task will require each time I/O processing with probability $1-\alpha$, and depart with probability α, we conclude that

$$P\{\text{task uses CPU } i \text{ times}\} = (1-\alpha)^{i-1}\alpha, \quad i = 1,2,... \qquad \text{(I.6.7-3)}$$

Therefore

$$\text{Avg. number of times that a task uses CPU processing} = \frac{1}{\alpha}\ ; \qquad \text{(I.6.7-4)}$$

$$\text{Avg. number of times that a task uses the I/O processor} = \frac{\alpha}{1-\alpha}\ . \qquad \text{(I.6.7-5)}$$

Hence, we obtain

$$\text{Avg. total CPU processing time required by a task} = (\alpha\mu_C)^{-1}\ [\text{sec}] \qquad \text{(I.6.7-6)}$$

$$\text{Avg. total I/O processing time required by a task} = \frac{1-\alpha}{\alpha\mu_I}\ . \qquad \text{(I.6.7-7)}$$

Let

$$P_n = P\{n \text{ tasks in the CPU, queued or being served}\}\ . \qquad \text{(I.6.7-8)}$$

Then, we obtain

$$P_n = \frac{1-\rho}{1-\rho^{N+1}}\ \rho^n\ , \quad n = 0,1,...,N \qquad \text{(I.6.7-9)}$$

where

$$\rho = \frac{\mu_I}{(1-\alpha)\mu_C} \qquad \text{(I.6.7-10)}$$

The average time delay (response time) D of a task in the system is obtained to be given by

$$D = \frac{N}{\alpha\mu_C}\,\frac{1-\rho^{N+1}}{\rho-\rho^{N+1}}\ . \qquad \text{(I.6.7-11)}$$

Note that the task response time D is equal to the sum of the task overall average waiting time in queues W and required overall average processing time. But,

$$\text{Avg. required task overall processing time} = \frac{1}{\alpha\mu_C} + \frac{1-\alpha}{\alpha\mu_I} \quad . \qquad \text{(I.6.7-12)}$$

Therefore, the overall average task waiting time W is

$$W = \frac{N}{\alpha\mu_C} \frac{1 - \rho^{N+1}}{\rho - \rho^{N+1}} - \left(\frac{1}{\alpha\mu_C} + \frac{1-\alpha}{\alpha\mu_I}\right) \quad . \qquad \text{(I.6.7-13)}$$

The CPU index of utilization U_C is given by

$$U_C = P\{\text{CPU is occupied}\} = 1 - P_0$$

$$= 1 - \frac{1 - \rho}{1 - \rho^{N+1}} = \frac{\rho - \rho^{N+1}}{1 - \rho^{N+1}} \quad . \qquad \text{(I.6.7-14)}$$

By (I.6.7-11) and (I.6.7-14) we conclude that the task response time D and the system utilization index U_C are related according to the formula

$$D = \frac{N}{\alpha\mu_C} \cdot U_C \quad . \qquad \text{(I.6.7-15)}$$

Relation (I.6.7-15) shows clearly how the task response time increases with the increase of the CPU utilization factor U_C. The above formulas need to be used in designing the system so that proper response-times and utilization values are attained.

More involved queueing models representing various interactions between a CPU (or several CPUs) and I/O devices can be developed using queueing network models. For the purpose of global performance prediction for the Space Shuttle computer network, the models presented in this section and the one presented in the next section are particularly useful.

I.6.8 An Analytical Model for the Computer System Performance Prediction

In studying the detailed behavior of the Space Shuttle computer system, one needs to model the interaction between the CPU and I/O queues. Such a simple cyclic-queue model, and its performance analysis, is presented in the previous section. This model can be used for a first-order study of the performance of the underlying computer system.

In this section a more involved cyclic queueing model is described and studied. This queueing model also incorporates the porper interaction between the CPU and I/O queues. The level of detail here is such that it allows the system engineer to study changes in hardware configurations and gross changes in the software.

The system model is shown in Figure I.6-7. Users, or sources, request for the processing of their associated jobs. Requests are first stored in queue 1, the queue for Main Storage, until space becomes available in main storage. After the job enters the main storage it actively competes for the use of the CPU or I/O devices. The job cycles between use of the CPU and I/O devices until it is completed. When a job is completed, a new job from the main storage queue replaces it. The source whose job is complete, sends a new request for job processing after a random think time delay.

The total number of jobs in the main storage and processing facility is limited to M.

Jobs are assumed to relinquish the CPU to carry out an I/O operation. We need to statistically characterize the length S_C

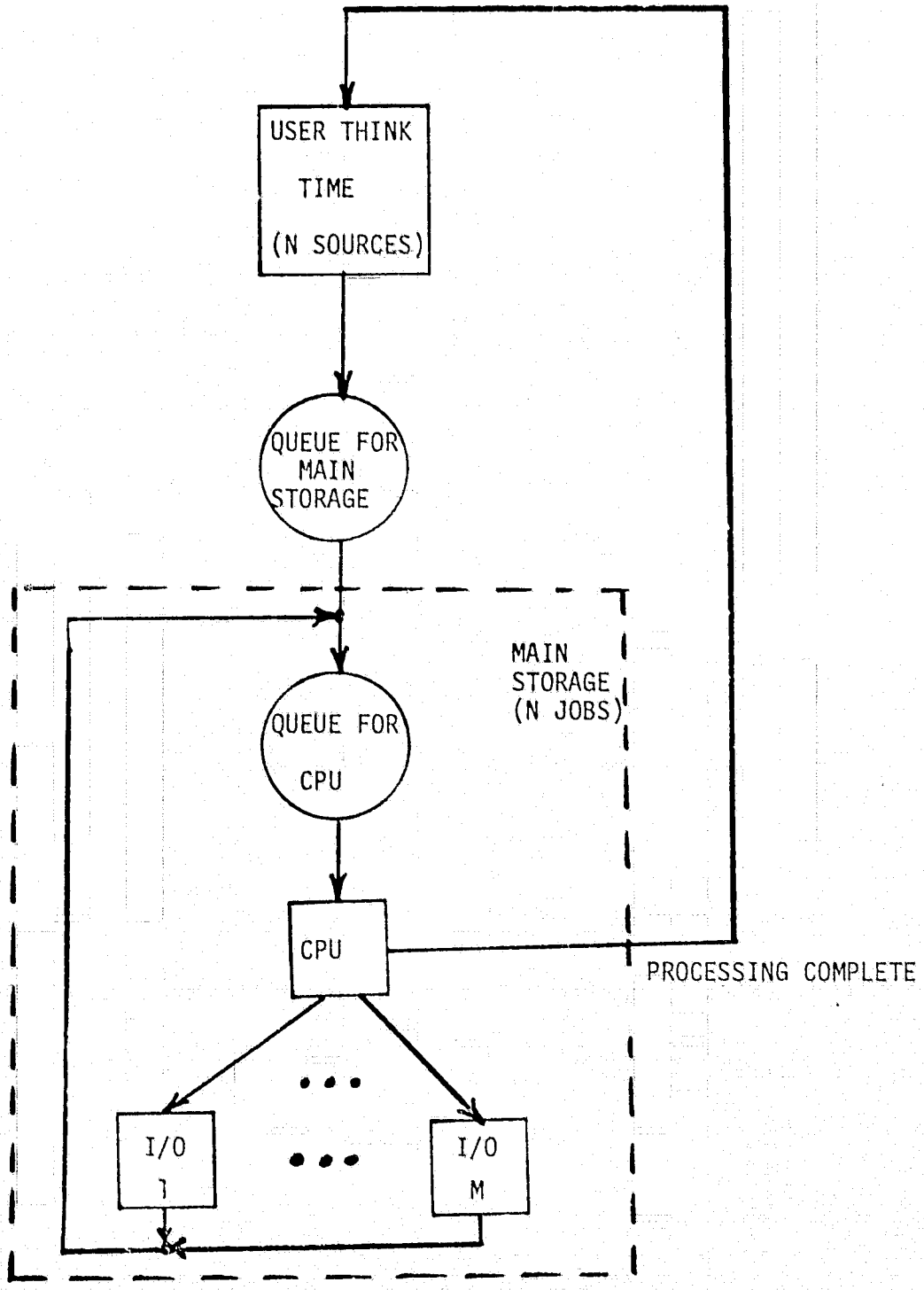
USER THINK
TIME
(N SOURCES)
QUEUE FOR
MAIN
STORAGE
MAIN
STORAGE
(N JOBS)
QUEUE FOR
CPU
CPU
PROCESSING COMPLETE
I/O
1
I/O
M

Figure I.6.7

of each CPU service period between successive I/O operations for those jobs in main storage. We assume here that either these periods are fixed length or they are random and exponentially distributed.

The I/O service time S_I will also be taken to be either of fixed length or random and exponentially distributed. It is also assumed here that no I/O queueing occurs. The I/O devices are taken to be identical processors operating as M parallel servers. A job requiring I/O processing will then be directed immediately to a free I/O device.

In studying the performance of this system, we assume that the system is sufficiently loaded so that there are always as many jobs requesting processing as the operating system will allow in main storage. Thus we take the number of jobs in the main storage system to be always M. Consequently, no more than N-M will be in think mode at any one time. This can be regarded as a fixed multiprogramming level M.

The model input parameters, reflecting the characteristics of the request traffic, required processing times, operating system and the hardware configuration, are summarized as follows.

τ_C = Average total CPU time required by a job

μ_C^{-1} = Average CPU time between I/O operations

μ_I^{-1} = Average service time for an I/O request

M = Number of jobs in the main processing system (level of multiprogramming)

N = Number of terminals (sources, users)

λ^{-1} = Average user think-time between requests

We also define

τ_I = Average total I/O time required by a job

K = Average number of times that a job requried I/O processing.

We then clearly obtain that

$$K = \frac{\tau_C}{\mu_C^{-1}} \quad , \tag{I.6.8-1}$$

and

$$\tau_I = K\mu_I^{-1} = \mu_I^{-1}\tau_C/\mu_C^{-1} \quad . \tag{I.6.8-2}$$

Note that the average time values τ_C, μ_C^{-1} include both the processing time of the job itself as well as the overhead time used by the system in running the job.

In studying such a computer system, the performance measures of interest are the following ones.

D = Job response time (sec)

= Average time delay of a job in the system from entry of request to completion of processing.

T = Computer system throughput (interactions/sec, or jobs served/sec)

= Average number of jobs departing from the system, per unit time, after their processing is completed.

U = CPU index of utilization

= Average fraction of time that the CPU is utilized for processing

= Probability that the CPU is occupied (busy, not idle).

The job response time D is obtained, in terms of the CPU index of utilization U, to be given by the formula

$$D = \frac{N\tau_C}{U} - \lambda^{-1} \quad [\text{sec}] \ . \qquad \text{(I.6.8-3)}$$

In terms of U, the system throughput is given by

$$T = \frac{U}{\tau_C} \quad [\text{interactions/sec}] \ . \qquad \text{(I.6.8-4)}$$

To prove and explain relations (I.6.8-3)(I.6.8-4) we note that following. Assume the system to run for a (long) period of τ sec. During this time assume that J jobs are processed, requiring a total time of τ_1 sec. Then

$$U = \frac{\text{CPU processing time}}{\text{Elapsed time}} = \frac{\tau_1}{\tau} = \frac{J\tau_C}{\tau} \ , \qquad \text{(I.6.8-5)}$$

$$T = \frac{\text{number of jobs served}}{\text{Elapsed time}} = \frac{J}{\tau} \ . \qquad \text{(I.6.8-6)}$$

Therefore

$$\frac{U}{T} = \tau_C \ , \qquad \text{(I.6.8-7)}$$

and eq. (I.6.8-4) results.

Observe again the system for a period of τ sec, during which J jobs are processed . During this time each of the N terminals is either in a response-time period (waiting for its job to complete processing) or in a think-time period (experiencing delay prior to the initiation of the next request). We have, during τ sec,

Average number of jobs completed per terminal = J/N . (I.6.8-8)

Hence,

Average time taken by a single interaction = $\frac{\tau}{J/N} = \frac{N\tau}{J}$ sec (I.6.8-9)

This time contains both system response time and user think-time. But

$$D = \frac{N\tau}{J} - \lambda^{-1} \quad . \qquad \text{(I.6.8-11)}$$

We observed in (I.6.8-5) that

$$\tau = \frac{J\tau_C}{U} \quad . \qquad \text{(I.6.8-12)}$$

Substituting (I.6.8-12) in (I.6.8-11), we derive (I.6.8-4).

Equations (I.6.8-3)-(I.6.8-4) thus allow to compute the message response time D and the system throughput T, once the CPU utilization index U is known. The latter is derived using the queueing techniques presented in previous sections. We obtain the following results.

For constant CPU and I/O service times, the CPU utilization index is given by

$$U = \begin{cases} \dfrac{M}{1+\mu_I^{-1}/\mu_C^{-1}} \quad , & \text{if } M \leq 1 + \dfrac{\mu_I^{-1}}{\mu_C^{-1}} \\ 1 \quad , & \text{if } M > 1 + \dfrac{\mu_I^{-1}}{\mu_C^{-1}} \end{cases} \qquad \text{(I.6.8-13)}$$

If we assume CPU and I/O service times to be exponentially distributed with means μ_C^{-1} and μ_I^{-1}, respectively, we obtain

$$U = 1 - \left\{ M! \sum_{n=0}^{M} \frac{1}{(M-n)!} \left(\frac{\mu_C^{-1}}{\mu_I^{-1}} \right)^n \right\}^{-1} \quad . \qquad \text{(I.6.8-14)}$$

Thus, Eqs. (I.6.8-3)-(I.6.8-4) and (I.6.8-13)-(I.6.8-14) yield the system performance measures D, T and U. This is expressed here in a rather simple form in terms of the major processing system and message-traffic characteristics.

In designing the system or modifying it to improve its performance or increase its capability or efficiency, one incorporates the given system parameters of interest and chooses the remaining ones to guarantee desired proper values for message delay D, CPU utilization U and system throughput T.

I.7 Queueing Modeling and Analysis Procedures for the Space Shuttle Orbiter Avionics System

I.7.1 The Queueing Model

The queueing model chosen to represent the Space Shuttle orbiter avionics system is described as follows. It is composed of the three system elements:

- The computer system.
- The data bus communication network.
- The application processes, user and destination terminals.

The combined model block diagram is shown in Fig. I.7-1.

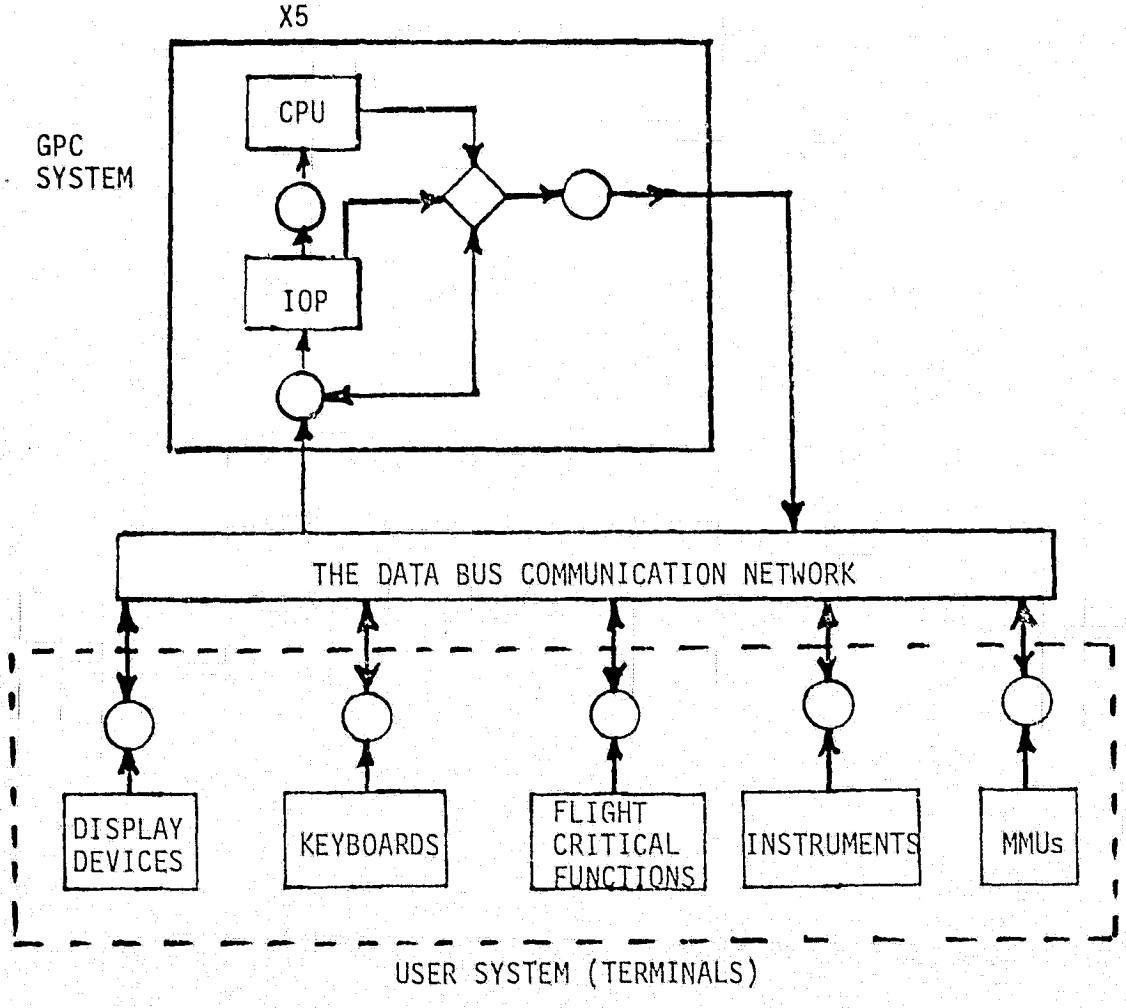

Figure I.7-1.

The computer model has been explained in Section I.2 (see Fig. I.2.1). We have combined here I/O and memory operations and accessing functions as single-unit I/O operations. The computer subsystem parameters are described in Section I.2.2.

The combination subnetwork structure has been outlined in Section I.1. The relevant parameters are presented in Section I.2.5. This network is composed of a set of half-duplex data buses properly shared by the computers. The buses are made available for information transmission or reception to the terminals at certain times.

The third subsystem is the user system. Terminals (users, tasks) are granted access to the communication network and the GPCs at certain times in accordance with their requests for service demands, TDM and polling processings, and GPC initiated actions.

The relevant system performance measures have been represented in Section I.3.

The heart of the system is the computer complex. We thus start by presenting queueing models for the computer system.

I.7.2 A Time Frame Model for the Computer System

We need to differentiate between cyclic and acyclic tasks. Such tasks have been statistically characterized in Sections I.2.3-I.2.4. Tasks for which computer time is reserved should also be described. Within the operational time period under consideration, we can thus make the following period definitions. We set:

T_F = duration of main time cycle (time frame) [sec]

T_D = duration of the time cycle period which is dedicated (reserved) to certain tasks (on a non-contention basis) [sec]

T_C = duration of the time cycle period which is used by cyclic tasks [sec]

T_A = duration of the time cycle period which is used by acyclic tasks [sec]

Thus, we have identified a time cycle (frame) of duration T_F. This frame is divided into the following three periods:

- The dedicated frame period, of duration T_D. This time period is reserved for certain tasks (application processes). These tasks can be cyclic or acyclic, scheduled or non-scheduled. During the period under consideration, the network controller assigns this periodic portion of the time frame, on a contention-free basis, to these tasks. Included are: scheduled tasks, routine updating tasks, routine information flow and processing duties, high priority dedicated services, etc.
- The cyclic frame period, of duration T_C. This period of time is periodically reserved for serving cyclic tasks. Service time portions within a cyclic frame period are assigned by the network controller (GPC) according to service demands (scheduled and unscheduled). These assignments are governed by the system priority service rules. A cyclic task which is assigned service time within a certain cyclic frame period, keeps the same assignment in succeeding cyclic frame periods, until its processing is completed, or until its service is pre-empted by the network controller.
- The acyclic frame period, of duration T_A. This period is used by acyclic tasks which arrive at random and require service time. Time is assigned in accordance with the system priority service discipline.

We clearly have (Fig. I.7.2)

$$T_F = T_D + T_C + T_A \quad . \tag{I.7.2-1}$$

In the operational period under consideration, tasks with dedicated service times (which total T_D sec) do not experience any waiting time. We can thus write

$$W_D(k) = 0 \ , \tag{I.7.2-2}$$

$$D_D(k) = S(k) \ , \tag{I.7.2-3}$$

where

$W_D(k)$ = waiting-time of a class k task with dedicated service

$D_D(k)$ = time delay of a class k task with dedicated service

$S(k)$ = overall GPC service time required by a class k message

Of course, the length of the period duration T_D assigned for dedicated service will affect the overall index of utilization of the computer system, as will be noted in the following analysis.

To obtain the delay-throughput performance characteristics of the computer system, we thus need to study the service of cyclic and acyclic tasks. This is carried out in the following sections.

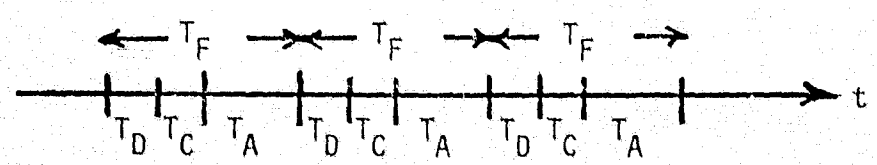

Figure I.7.2.

I.7.3 Queueing Analysis for Cyclic Tasks: Model I

We consider, in these sections, cyclic tasks which are served during the cyclic frame periods. Each cyclic frame period is of length T_C sec. Any two consecutive cyclic frame periods are separated by a time period of duration

$$T_A + T_D = T_F - T_C \text{ sec} . \qquad (I.7.3\text{-}1)$$

Assume that the computer system can serve N_C cyclic tasks during each cyclic frame period. For simplicity of presentation, we also assume that equal service times are assigned to all served cyclic tasks, during each cyclic period. Therefore, a served cyclic task is granted to a fixed service time of duration Δ sec, where

$$\Delta = T_C/N_C . \qquad (I.7.3\text{-}2)$$

Assume cyclic tasks to arrive at the system according to a Poisson process with intensity

$$\text{Arrival intensity} = \lambda_C \text{ [cyclic tasks/sec]} . \qquad (I.7.3\text{-}3)$$

Each cyclic task is assumed to require a service time S_C which is exponentially distributed with a mean (required computer) service time equal to

$$E(S_C) = \mu_C^{-1} \text{ [sec/cyclic task]} . \qquad (I.7.3\text{-}4)$$

When any one of the N_C time slots during a cyclic period becomes available, upon the termination of service a cyclic task, it can be assigned to any one of the queued cyclic tasks waiting for service. The queueing system under consideration thus becomes

an N_C-server queueing system. However, it is not a regular multi-server queueing system, since it experiences interruptions in service. After s service period of T_C sec., the service granted to cyclic tasks is interrupted for a period of T_F-T_C sec. Subsequently, service is resumed (simultaneously given to N_C cyclic tasks) for another period of T_C sec., and then interrupted again, and so on.

A proper simple approximate technique for the performance analysis of this interrupted multi-server queueing model is developed here and described in the following. We consider an equivalent non-interruptable queueing system with N_C servers and the following parameters. To incorporate the original interruption times, we let the equivalent service demand S_C' be exponentially distributed with mean service time

$$E(S_C') = \mu^{-1} T_F/T_C \quad . \qquad \text{(I.7.3-5)}$$

The arrival intensity remains equal to λ_C [cyclic tasks/sec].

Considering this equivalent queueing model, we perform the associated queueing analysis and obtain the following results (in accordance with the formulas presented in Section I.6.6).

By this model, we assume that each served task is processed by the computer system for a period of Δ sec, during each T_C sec cycle. A number of N_C cyclic tasks are served simultaneously. Each task will thus require an average of

Avg. No. cyclic periods used by a cyclic task

$$= \frac{\mu^{-1}}{\Delta} = \frac{\mu^{-1} N_C}{T_C} = E(S_C')\ (N_C/T_F) \qquad \text{(I.7.3-6)}$$

The queueing analysis follows the procedure described in Section I.6.6, when (I.7.3-5) is incorporated. The following results

are subsequently obtained.

Let

$$P_n = P\{\text{n cyclic tasks in the system, queueing or being served}\} \quad . \tag{I.7.3-7}$$

Define the traffic intensity parameter ρ by

$$\rho = \frac{\lambda T_F}{N_C \mu T_C} \quad . \tag{I.7.3-8}$$

For the system queueing process to be <u>stable</u>, so that queue-sizes and task response time would not become arbitrarily high, we must require

$$\rho = \frac{\lambda T_F}{N_C \mu T_C} < 1 \quad . \tag{I.7.3-9}$$

Henceforth, relation (I.7.3-9) is assumed to hold. Then,

$$P_0 = \left\{ \frac{(N_C\rho)^{N_C}}{N_C!(1-\rho)} + \sum_{i=0}^{N_C-1} \frac{(N_C\rho)^i}{i!} \right\}^{-1} ; \tag{I.7.3-10}$$

and

$$P_n = \begin{cases} \dfrac{(N_C\rho)^n}{n!} P_0 , & \text{if } n > N_C \\ \dfrac{N_C^{N_C}}{N_C!} \rho^n P_0 , & \text{if } n \geq N_C \end{cases} \tag{I.7.3-11}$$

The GPC index of utilization U_C (see definition by Eq. (I.3.1-1)) is therefore given by

$$U_C(C) = P\{\text{a GPC is busy in processing a cyclic task during the cyclic period}\}$$
$$= 1 - P_0 \quad , \tag{I.7.3-12}$$

where P_0 is given by (I.7.3-10).

The GPC throughput in processing cyclic tasks (see definitions (I.3.1-17)-(I.3.1-18)), assuming none to be rejected, is given by

$TTH_C(C)$ = The GPC cyclic task throughput

= Average number of cyclic tasks processed by the GPC per sec

= λ_C cyclic tasks/sec . (I.7.3-13)

To obtain the throughput in bps, we set

C = GPC average service rate in bps . (I.7.3-14)

Then,

$TH_C(C)$ = GPC throughput in bps for cyclic tasks

$= \lambda_C \mu^{-1} C$ bps (I.7.3-15)

The average task waiting time for a cyclic task is equal to

Average Cyclic Task Waiting-Time

$$= W_C = \frac{\mu^{-1} T_F}{T_C N_C} \left[1 - \sum_{n=0}^{N_C-1} n P_n \right] , \quad \text{(I.7.3-16)}$$

where P_n is given by (I.7.3-10)(I.7.3-11).

The average cyclic task time-delay, response time, D_C, is thus given by

D_C = average cyclic task response time

$= W_C + E(S_C')$

$$= \frac{\mu^{-1} T_F}{T_C} + \frac{\mu^{-1} T_F}{T_C N_C} \left[1 - \sum_{n=0}^{N_C-1} n P_n \right] . \quad \text{(I.7.3-17)}$$

The variance and distribution of the task response time are obtained similarly.

If a finite source model is desired, the proper formulas follow by Eqs. (I.6.6-8)-(I.6.6-10).

The study of buffer overflow characteristics is illustrated by the following model. We let (see also Section I.2.4)

M_C = overall (average) storage capacity for cyclic tasks (I.7.3-18)

Assume that: $M_C > N_C$. Thus, assume that no more than an overall number of M_V cyclic tasks can be stored in the system. Then, using the queueing models and methods of Section I.6.6 we obtain the queue-size probabilities:

$$P_0 = \left\{ \sum_{k=0}^{N_C-1} \frac{(N_C\rho)^k}{k!} + \sum_{k=N_C}^{M_C} \frac{N_C^{N_C}}{N_C!} \rho^k \right\}^{-1} ; \qquad \text{(I.7.3-19)}$$

$$P_n = \begin{cases} \dfrac{(N_C\rho)^n}{n!} P_0 , & \text{if } n < N_C , \\ \dfrac{N_C^{N_C}}{N_C!} \rho^n P_0 , & \text{if } N_C \le n \le M_C , \\ 0 , & \text{if } n > M_C \end{cases} \qquad \text{(I.7.3-20)}$$

The probability of overflow is subsequently given by

$$POF = P_{M_C} = \frac{M_C^{M_C}}{(M_C)!} \rho^{M_C} P_0 , \qquad \text{(I.7.3-21)}$$

where P_0 is given by (I.7.3-19) and ρ is given by (I.7.3-8). For this system, with a limited storage capacity, it is not necessary any more to require $\rho < 1$.

The GPC index of utilization and response time are given again according to formulas (I.7.3-12) and (I.7.3-17), with P_0 now expressed by Eq. (I.7.3-19).

Substituting the proper system and task-traffic parameters, as well as the parameters characterizing the mission phase under consideration, the above formulas allow us to compute the system performance indices, related to the service of cyclic tasks

I.7.4 Queueing Analysis for Cyclic Tasks: Model II

To derive at a more detailed GPC queueing model, in describing the service of cyclic tasks, we can use the models described in Sections I.6.7 and I.6.8. Using these models, we can describe the CPU/IO processing interactions in the GPC system.

Assume thus the GPC service system to be described by the closed loop model illustrated by Fig. I.6.6.

We assume that the number of cyclic tasks in the GPC is kept constant, equal to N_C, as in the previous section.

A task entering the GPC system joins the CPU queue. A task can enter GPC service only when a previous one has been completed, assuming thus a constant number of N_C cyclic tasks in the system. After receiving service by the CPU a cyclic task leaves the system with probability α_c. With probability $1-\alpha_c$ this task will subsequently enter the I/O queue. There, tasks are served on a first-come first-served basis. Upon departure from the I/O processor, a task joins immediately the CPU queue.

We assume each cyclic task to require CPU and I/O processing times which are i.i.d. exponentially distributed random variables with means

Avg. CPU service time required by a cyclic task

$$= \mu_C^{-1}(C) \text{ [sec]} \qquad \text{(I.7.4-1)}$$

Avg. I/O service time required by a cyclic task

$$= \mu_I^{-1}(C) \text{ [sec]} \qquad \text{(I.7.4-2)}$$

We obtain that

Avg. number of times that a cyclic task uses the I/O processor $= \dfrac{\alpha_C}{1-\alpha_C}$. (I.7.4-3)

Also,

Avg. total CPU processing time required by a cyclic task

$$= [\alpha_C \mu_C(C)]^{-1} \text{ [sec]} \; ; \qquad \text{(I.7.4-4)}$$

Avg. total I/O processing time required by a cyclic task

$$= \frac{1-\alpha_C}{\alpha_C \mu_I(C)} \quad \text{[sec]} \qquad \text{(I.7.4-5)}$$

We define the queue size probabilities

P_n = P{n cyclic tasks in the CPU, queued or being served}. (I.7.4-6)

To perform the queueing analysis we note again that the service of a cyclic task by the GPC system proceeds in an interrupted periodic manner. We perform an approximate queueing analysis by setting the effective mean CPU and I/O processing times, denoted as $\tilde{\mu}_C^{-1}(C)$ and $\tilde{\mu}_I^{-1}(C)$, respectively, to be

$$\tilde{\mu}_C^{-1}(C) = \mu_C^{-1}(C) \; \frac{T_F}{T_C} \; ; \qquad \text{(I.7.4-7)}$$

$$\tilde{\mu}_I^{-1}(C) = \mu_I^{-1}(C) \; \frac{T_F}{T_C} \; ; \qquad \text{(I.7.4-8)}$$

following the same approximation adopted in the previous section. The following analysis results are obtained (see also Section I.6.7).

The system traffic intensity parameter ρ is set equal to

$$\rho = \frac{\mu_I(C)}{(1-\alpha_C)\mu_C(C)} \quad . \tag{I.7.4-9}$$

We obtain the queue-size probability P_n to be given by

$$P_n = \frac{1-\rho}{1-\rho^{N_C+1}} \rho^{N_C} \quad , n = 0,1,\ldots,N_C \quad . \tag{I.7.4-10}$$

The average response time (time delay) D_C of a cyclic task in the system is obtained to be expressed as

$$D_C = \frac{N_C T_F}{\alpha_C \mu_C(C) T_C} \quad \frac{1-\rho^{N_C+1}}{\rho-\rho^{N_C+1}} \quad . \tag{I.7.4-11}$$

The average task waiting time W_C is

$$W_C = D_C - \left[\frac{T_F}{\alpha_C \mu_C(C) T_C} + \frac{(1-\alpha_C)T_F}{\alpha_C \mu_I(C) T_C} \right] \tag{I.7.4-12}$$

The CPU index of utilization is equal to

$U_{CPU}(C)$ = CPU index of utilization by cyclic tasks

= P{CPU is occupied by cyclic tasks during the cyclic period}

$$= (1-P_0) = \frac{\rho-\rho^{N_C+1}}{1-\rho^{N_C+1}} \quad . \tag{I.7.4-13}$$

We note that the cyclic task response time D_C and the CPU index of utilization $U_{CPU}(C)$ are related by the formula

$$D_C = \frac{N_C T_F}{\alpha_C \mu_C(C) T_C} \; U_{CPU}(C) \quad . \qquad \text{(I.7.4-14)}$$

Thus, we can use these formulas to compute, for each mission phase, the relevant performance indices.

I.7.5 Queueing Analysis for Cyclic Tasks: Model III

Model III for the GPC service system is chosen to be the model described in section I.6.8 (see Fig. I.6.7). See Section I.6.8 for detailed description and derivations. All the system parameters used are denoted as in this section, with the following modifications.

We consider here only cyclic tasks. Subseqeuntly, parameters are denoted as: $\tau_C(C)$, $\mu_C^{-1}(C)$, $\mu_I^{-1}(C)$, λ_C^{-1}, $\tau_I(C)$,$K(C)$.

We set $M=N_C$ to denote the maximal number of cyclic tasks in the main processing system.

We set $N=N(C)$ to denote the number of cyclic terminals (sources, users). The average terminal thinking time is λ_C^{-1}.

Rederiving the delay-throughput expressions for the present case, we obtain the cyclic task response time D_C to be given as a function of the CPU index of utilization by cyclic tasks during the cyclic period, $U_{CPU}(C)$,

$$D_C = \frac{N_C \tau_C(C) T_F}{U_{CPU}(C) T_C} - \lambda_C^{-1} \quad \text{[sec]} \qquad \text{(I.7.5-1)}$$

The throughput is given by

$$TH_C = \frac{U_{CPU}(C)}{\tau_C(C)} \quad \text{[interactions/sec]} \qquad \text{(I.7.5-2)}$$

For exponentially distributed CPU and I/O service times, we obtain the CPU index of utilization to be equal to

$$U_{CPU}(C) = 1 - \left\{ N_C! \sum_{n=0}^{N_C} \frac{1}{(N_C-n)!} \left(\frac{\mu_C^{-1}(C)}{\mu_I^{-1}(C)} \right)^n \right\}^{-1} \quad \text{(I.7.5-3)}$$

Using these formulas, one computes the system indices of performance when considering the service of cyclic tasks, under various mission conditions.

Using Eq. (I.7.5-3), one computes the CPU index of utilization $U_{CPU}(C)$. The latter indicates the fraction of the cyclic frame period that is occupied by the service of cyclic tasks. The parameters involved in this computation are:

N_C = maximal number of cyclic tasks served during a single cyclic frame period

$\mu_C^{-1}(C)$ = average CPU time for cyclic tasks between I/O operations, during the cyclic period

$\mu_I^{-1}(C)$ = average service time for an I/O cyclic request, during cyclic period

The computer system throughput for cyclic tasks, TH_C is evaluated by using Eq. (I.7.5-2). It yields the average number of cyclic task completions per unit time within the cyclic frame periods.

Finally, the response-time (average time delay) of a cyclic task, D_C, is computed by using Eq. (I.7.5-1). It yields the average time delay of a cyclic task, from the instant it indicates its task request to the instant its service is completed.

The additional parameters involved in Eq. (I.7.5-1)-(I.7.5-2) are:

$\tau_C(C)$ = average total CPU time required by a cyclic task

λ_C^{-1} = average think time between initiation of a new cyclic task and the completion of the previous one

T_C = duration of a cyclic frame period

T_F = duration of a frame period (main system cycle)

I.7.6 Queueing Analysis for Acyclic Tasks: Priority Model I

We consider now the service of acyclic tasks. Requests for service by such tasks arrive at random, according to the statistics of a Poisson process with intensity λ_A acyclic tasks/sec. Thus:

Average number of new acyclic tasks (requests for service) arrivals = λ_A acyclic tasks/sec . (I.7.6-1)

The computer system can serve acyclic tasks only during the acyclic frame periods (see Section I.7.2). Therefore, acyclic tasks are served by the GPC during their period for a length of time of T_A sec; then, service is interrupted for T_F-T_A sec; subsequently, service resumes for another T_A = sec, and so on.

We wish to describe here a simple queueing model for the GPC service system, which incorporates different task priorities (see Section I.5). We consider a generalization of the priority queueing model described and analyzed in Section I.5.4.

Acyclic tasks are classified into p priority classes. A class-k task is a task with priority number k, k = 1,2,...,p. Class-1 tasks attain the highest priority, while class-p tasks have the lowest priority.

Under a nonpreemptive service discipline, when computer service time becomes available, a class-i task is served before any class-j

task if i < j. Within each class, tasks are served in order of arrival. No preemption (interruption) of any task service is allowed.

Under a preemptive resume service discipline, class-i tasks are again preefered over class-j tasks if i < j, as above. However, now we allow the preemption (interruption of service) of a lower priority task when a higher priority task arrives at the system.

We assume that class-k acyclic tasks arrive at the system according to a Poisson process with intensity $\lambda_A(k)$ tasks/sec:

Intensity of arrival of priority-k acyclic tasks

$$= \lambda_A(k) \text{ tasks/sec}, \quad k = 1,2,\ldots,p; \tag{I.7.6-2}$$

so that

$$\lambda_A = \sum_{k=1}^{p} \lambda_A(k) \quad . \tag{I.7.6-3}$$

We set

$S_A(k)$ = GPC processing time required by a class k acyclic task

The corresponding required service time moments are

$$\overline{S}_A(k) = E\{S_A(k)\} = \text{mean service time for priority-k task;} \tag{I.7.6-4}$$

$$\overline{S_A^2}(k) = E\{S_A^2(k)\} \quad . \tag{I.7.6-5}$$

In particular, we note that if an acyclic class-k task required GPC service time is exponentially distributed with mean $\mu_A^{-1}(k)$, then

$$\overline{S}_A(k) = \mu_A^{-1}(k) \quad , \quad \overline{S_A^2}(k) = \mu_A^{-2}(k) \quad . \tag{I.7.6-6}$$

On the other hand, if each k-class task has a fixed service

requirement, $S_A(k) = \mu_A^{-1}(k)$, then

$$\overline{S}_A(k) = \mu_A^{-1}(k), \quad \overline{S_A^2}(k) = \mu_A^{-1}(k) \quad . \tag{I.7.6-7}$$

We set

$$\rho_i = \lambda_i \overline{S}_A(i) \frac{T_F}{T_A} \quad , \quad i = 1,2,\ldots,p; \tag{I.7.6-8}$$

$$\sigma_j = \sum_{i=1}^{j} \rho_i \quad ; \tag{I.7.6-9}$$

$$\rho = \sigma_p = \sum_{i=1}^{p} \rho_i \quad . \tag{I.7.6-10}$$

For queue-size stability (so that queue-sizes and message delay would not become arbitrarily high) we requrie

$$\rho < 1 \quad . \tag{I.7.6-11}$$

We set

$W_A(k)$ = average waiting-time for a priority-k acyclic task

$D_A(k)$ = average time delay (response time) for a priority-k acyclic task

$X_A(k)$ = average queue-size of priority-k acyclic tasks

X_A = average queue-size of all acyclic tasks.

Assume first a nonpreemptive service discipline. The same approximation for describing the service interruption used in previous sections is employed. We obtain the following formulas.

$$W_A(k) = \frac{A}{2(1-\sigma_k)(1-\sigma_{k+1})} \quad , \; k = 1,2,\ldots,p \; , \tag{I.7.6-12}$$

where

$$A = \sum_{i=1}^{p} \lambda_A(i) \left(\frac{T_F}{T_C}\right)^2 \overline{S_A^2}(i) \quad ; \qquad \text{(I.7.6-13)}$$

$$D_A(k) = W_A(k) + \overline{S}_A(k) \quad ; \qquad \text{(I.7.6-14)}$$

$$X_A = \frac{\lambda_A A}{2(1-\rho)} \quad .$$

The system index of utilization is:

U_A = index of utilization of GPC by acyclic tasks

= P{GPC is occupied by acyclic tasks during the acyclic frame periods} .

It is given by

$$U_A = \rho = \sum_{i=1}^{p} \lambda_A(i) \frac{T_F}{T_A} \overline{S}_A(i) \quad . \qquad \text{(I.7.6-15)}$$

If a preemptive-resume service discipline is assumed, we obtain the following formulas.

$$D_A(k) = \frac{B_k}{2(1-\sigma_k)(1-\sigma_{k+1})} \quad , \qquad \text{(I.7.6-16)}$$

where

$$B_k = 2(1-\sigma_k)\overline{S}_A(k) \frac{T_F}{T_C} + \sum_{i=1}^{k} \lambda_A(i) \left(\frac{T_F}{T_C}\right)^2 \overline{S_A^2}(i) \quad ; \qquad \text{(I.7.6-17)}$$

$$X_A(k) = \lambda_A(k) D_A(k) \quad ; \qquad \text{(I.7.6-18)}$$

$$X_A = \sum_{k=1}^{p} X_A(k) \quad . \qquad \text{(I.7.6-19)}$$

Thus, under a preemptive resume priority service discipline the message response time is given by Eqs. (I.7.6-16)-(I.7.6-17), while the queue-size are given by Eqs. (I.7.6-18)-(I.7.6-19). We note that the required average buffer sizes are estimated by the queue-size values of (I.7.6-18)-(I.7.6-19). The computer index of the utilization is expressed again by Eq. (I.7.6-15).

These formulas, and their extensions, as outlined in this report, allow us to analyze the computer system performance under the proper mission conditions.

We have demonstrated here the use of a simple priority queueing model. Other priority queueing models have been presented and analyzed in Sections I.4 and I.5. A multitude of time-sharing queueing models are presented in Section I.4. Various priority queueing models are discussed and investigated in Section I.5. The results presented there are directly applicable to the queueing modeling and analysis of the Space Shuttle avionics computer system studied here. The only modification necessary, when considering acyclic tasks, is the incorporation of an effective required service time equal to $S_A(k) \frac{\tau_F}{\tau_A}$.

In this way, the proper traffic, task and subsystem models and parameters, presented in Section I.2, are used to evaluate the computer system performance measures presented in Section I.3. The results of Sections I.4-I.5 are properly integrated.

I.7.7 Queueing Analysis for Acyclic Tasks: Models II

Queueing models describing the service of acyclic tasks, while detailing the CPU/IO interactions are developed and studied in a manner which is completely analogous to those presented in

Sections I.7.4-I.7.5. The only differences lie in:

- Choosing system service and arrival parameters for acyclic tasks, rather than cyclic tasks;
- Choosing the proper number N (rather than N_C) for the maximal number of tasks allowed simultaneously to be in GPC service;
- Replacing T_F/T_C by T_F/T_A,

Otherwise, we obtain the same relationships for the computer system indices of performance.

I.7.8 Joint Queueing Analysis

The results in Sections I.7.3-I.7.7 are combined as follows to yield the indices of performance for the global computer system.

The response-time (average message delay) of a cyclic and an acyclic task is given by D_C and D_A, respectively. If priority-k tasks are considered, the corresponding response times are $D_C(k)$ and $D_A(k)$. The proper formulas are given in Sections I.7.3-I.7.7. The time-frame division between dedicated, cyclic and acyclic periods has been exposed in Section I.7.2.

The traffic intensities of dedicated, cyclic and acyclic tasks are denoted as λ_D,λ_C, and λ_A, respectively. Then, if we choose a certain task at random, its average queueing delay (response-time) $\overline{D}$ will be equal to

$$\overline{D} = \lambda^{-1}\{\lambda_D D_D + \lambda_C D_C + \lambda_A D_A\} , \qquad (I.7.8\text{-}1)$$

where

$$\lambda = \lambda_D + \lambda_C + \lambda_A . \qquad (I.7.8\text{-}2)$$

The function D_D denotes the average delay of a dedicated task.

For such a task we have presently reserved computer time. We can thus assume its waiting time to be equal to 0, and set its response time equal to its average required service duration. We set

$\bar{S}_D$ = average required computer service time for a dedicated task.

Subsequently, the dedicated task response-time is equal to

$$D_D = \bar{S}_D \frac{T_F}{T_D} \qquad \text{(I.7.8-3)}$$

In computing the computer system queue-sizes, we write

$$X = X_D + X_A + X_C \ , \qquad \text{(I.7.8-4)}$$

where

X = global system average queue-size;

X_D = average queue-size of dedicated tasks,

X_A = average queue-size of acyclic tasks,

X_C = average queue-size of cyclic tasks.

If we assume that presently no dedicated tasks are waiting, as noted above, then the queue-size X_D is equal to the number of dedicated tasks presently being in service.

The GPC index of utilization U is computed as follows. We have

U_C = GPC index of utilization for cyclic tasks in the cyclic periods,

U_A = GPC index of utilization for acyclic tasks in acyclic periods,

U_D = GPC index of utilization for dedicated tasks.

The indices U_C and U_A have been computed in Sections I.7.3 - I.7.7. The index U_D is set to be

U_D = fraction of time that the dedicated frame period (of length T_D) is used. (I.7.8-5)

Function U_P is determined by the state of the mission as pertaining to how much dedicated service is presently required.

The GPC index of utilization U, in serving all these three classes of tasks, is given by

U = fraction of time the GPC is idle

= P{GPC not occupied in serving any dedicated, or cyclic, or acyclic task}. (I.7.8-6)

We conclude that

$$U = 1 - (1-U_D)(1-U_C)(1-U_A) \ . \qquad \text{(I.7.8-7)}$$

Using the index utilization formulas presented in previous sections, we can determine the time frame values T_D, T_A, T_C, that will yield the proper desired high (and even maximum) system utilization values, under proper task response-time and queue-size constraints. The system designer and analyst can thus deduce, adjust and plan the proper compromised system performance values.

I.7.9 Queueing Analysis for User Terminals: Output Traffic

The queue-size behavior of a user terminal is described by the following model.

We describe the process of transmission of requests or messages from a user terminal to the computer complex by a cyclic polling TDM (time-division multiplexing) procedure. For that purpose

we divide messages into fixed-length data units called packets. A packet will contain an average of μ^{-1} bits:

$$\text{Average packet length} = \mu^{-1} \text{ bits} . \qquad \text{(I.7.9-1)}$$

A packet can contain request for service information or any data information transmitted to the computer system.

Data is transmitted across the data-bus network at a rate of C bps:

$$\text{Transmission rate} = C \text{ bps} \qquad \text{(I.7.9-2)}$$

For the avionics network, we have

$$C = 10^6 \text{ bps}.$$

Subsequently, the packet transmission time is equal to τ sec, where

$$\tau = (\mu C)^{-1} \text{ [sec]} . \qquad \text{(I.7.9-3)}$$

Assume now that we establish a basic time slot duration sec, so that the terminal under consideration is polled as follows. It is assigned, on a fixed TDM basis, a single slot for information transmission, once every M slots. Thus, the terminal can transmit a packet of information in its assigned slot of τ sec duration; subsequently, it has to wait $(N-1)\tau$ sec for its next assigned slot to occur, and so on (see Fig. I.7.2).

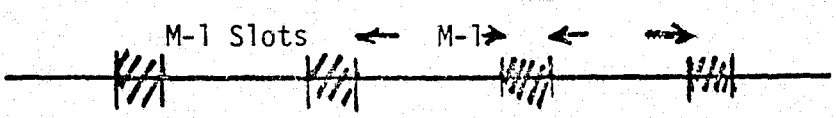

Figure I.7.2

Assume now that the terminal generates packets (of service requests or applications data) according to a Poisson process with intensity λ_p packets/sec. Thus

$$\text{Packet intensity at a terminal} = \lambda_p \text{ packets/sec} \qquad \text{(I.7.9-4)}$$

Let the terminal indices of performance be given by:

D_p = average delay (response-time) for a packet at a user terminal [sec], (I.7.9-5)

X_p = average queue-size (in packets) at user buffer, (I.7.9-6)

U_p = index of utilization of a user terminal buffer. (I.7.9-7)

We proceed with a TDMA queueing theoretical analysis and obtain the following results for the terminal performance functions.

$$D_p = \frac{1}{2} M + 1 + \frac{M\rho}{1-\rho} \quad , \qquad \text{(I.7.9-8)}$$

where

$$\rho = M\lambda_p < 1 \; ; \qquad \text{(I.7.9-9)}$$

$$X_p = \lambda_p D_p \; ; \qquad \text{(I.7.9-10)}$$

$$U_p = \rho = M\lambda_p \; . \qquad \text{(I.7.9-11)}$$

Thus, in observing the queueing characteristics of user transmissions and its buffer, as reflected by eqs. (I.7.9-9)-(I.7.9-11) we deduce the following conclusions. The packet delay and buffer queue-size increases rapidly as U_p approaches its maximal allowable value of 1. Fixing an average allowable queue size value X_p, to yield an acceptable probability of overflow (POF) value, results

by (I.7.9-10) with a delay value $D_p = \lambda_p^{-1} X_p$, if we assume an input rate equal to λ_p. The delay function D_p is related to $U_p = \rho = M\lambda_p$ and M by Eq. (I.7.9-8). We subsequently solve for the associated value of M. The latter specifies the required frequency of polling (equal to $\frac{1}{M}$) for this terminal.

We have presented here a model that can apply to the multitude of terminals, users, subsystems and application processes in the Space Shuttle avionics system. Time-sharing and priority queueing models, presented in the previous sections can also be applied.

I.7.10 Queueing Analysis for User Terminals: Input Traffic

We consider in this section the terminal buffer queue-size behavior in terms of messages arriving at the terminal from the computer system.

Consider a specific terminal where messages arrive from the computer complex according to a Poisson process with a rate of

λ_t = message arrival rate at a terminal [mess./sec] (I.7.10-1)

Each message is assumed to contain S_t bits/mess. Thus:

$\overline{S_t} = E(S_t)$ = mean terminal message length [bits/sec] (I.7.10-2)

$\overline{S_t^2} = E(S_t^2)$ (I.7.10-3)

The terminal is assumed to process (and absorb) the received information at a rate of

C_t = terminal processing rate [bps] . (I.7.10-4)

Subsequently, each message requires a terminal processing time of

$S_t C_t^{-1}$ [sec/mess.] .

The performance indices of interest are:

X_t = average message queue-size in terminal buffer

D_t = average delay of a message in terminal buffer

U_t = index of utilizationof terminal buffer.

Regarding the terminal service system in processing input data from the computer as a single-server queueing system, we obtain the following results (see Section I.4.3)

$$D_t = \frac{\lambda_t \overline{S_t^2} C_t^{-2}}{2(1-\rho)} + \overline{S}_t C_t^{-1} \quad , \tag{I.7.10-5}$$

where

$$\rho = \lambda \overline{S}_t C_t^{-1} < 1 \; ; \tag{I.7.10-6}$$

$$X_t = \lambda_t D_t = \frac{\lambda_t^2 \overline{S_t^2} C_t^{-2}}{2(1-\rho)} + \rho \; ; \tag{I.7.10-7}$$

$$U_t = \rho = \lambda \overline{S}_t C_t^{-1} \quad . \tag{I.7.10-8}$$

If message lengths are exponentially distributed with mean μ_t^{-1} [bits/mess.], we have

$$\overline{S}_t = \mu_t^{-1} C_t^{-1} \; ; \quad \overline{S_t^2} = \mu_t^{-2} C_t^{-2} \quad . \tag{I.7.10-9}$$

For exponentially distributed message lengths (I.7.10-9), we can also derive the performance measure while assuming a buffer with finite capacity of

$$L_t = \text{terminal buffer capacity (in number of messages).} \tag{I.7.10-10}$$

Using this the results presented in Section I.6.4, we conclude the following expressions.

$$X_t = \frac{1-\rho}{1-\rho^{L_t+1}} \sum_{n=0}^{L_t} n\rho^n , \qquad \text{(I.7.10-11)}$$

where

$$\rho = \lambda_t \bar{S}_t C_t^{-1} = \lambda_t \mu_t^{-1} C_t^{-1} \; ; \qquad \text{(I.7.10-12)}$$

$$U_t = \frac{(1-\rho^{L_t})}{1-\rho^{L_t}}$$

$$D_t = \mu_t^{-1} + \frac{\mu_t^{-1}}{1-P_R} X_t , \qquad \text{(I.7.10-13)}$$

where

$$P_R = (1-\rho) \frac{\rho^{L_t}}{1-\rho^{L_t+1}} \qquad \text{(I.7.10-14)}$$

An additional important measure of performance is now expressed by the probability that the buffer is saturated (overflow), POF_t. This is also equal to the probability that an arriving message is rejected (not accepted) at the terminal, denoted as P_R, due to buffer flow. We have:

POF_t = probability of terminal buffer overflow

= P_R = probability of message rejection

$$= (1-\rho) \frac{\rho^{L_t}}{1-\rho^{L_t+1}} \qquad \text{(I.7.10-15)}$$

Thus, in designing and analyzing the terminal system we specify and compute the delay, utilization and POF measures, using the performance formulas given above. Other time-sharing and priority queueing models can be applied and analyzed, following the presentations and results presented in the previous sections.

I.8 SYNCHRONIZATION METHODS FOR THE DATA PROCESSING SYSTEM

I.8.1 Synchronization Considerations for the Data Processing System

Due to the distributed control of redundant sensors among the Space Shuttle avionics network computers, an unacceptable time skew can exist between redundant inputs unless the GPCs are synchronized prior to initiating the inputs. Similarly, unacceptable data-skew may exist at the voting effectors unless a synchronization procedure is employed prior to initiating outputs. In addition, unacceptable command differences may exist at the voting effectors unless synchronization occurs at proper states during program execution.

Synchronizationis accomplished in the Space Shuttle computer complex by using inter-computer discrete signals and synchronization software.

Program synchronization is required as well, since computers that do not use exactly the same data for computing flight-control outputs experience command divergence effects. The time required to synchronize program execution depends on the design of the flight software operating system. A fixed time-slice system (in which all processes are run within a given cycle time) requires a single synchronization point in each computational cycle. An interrupt-driven system must synchronize at all points at which data are calculated in one process and used in another, and at all points needed to preserve identical process sequences in all computers of the set.

Synchronization requirements between the GPCs also arise due to error detection and recovery objectives. To provide a smooth switchover in the case of a failure, the computers must possess

some degree of synchronization even if the synchronization implementation uses only the intercomputer communication lines.

To achieve a high degree of error detection, comparison and voting procedures need to be employed. This requires the outputs of the GPCs feeding the comparison/voting stage to be synchronized.

- A software initiated synchronization is performed before:
- Input commands are issued;
- Outputs are exchanged for comparison purposes;
- Compool is updated by the background to pass information to the foreground;
- Real time is obtained.

A list of all active output must be maintained, for comparison or voting purposes. For a "bit-by-bit" comparison shceme to perform satisfactorily, all inputs to the computers must be identical. These include sensor inputs, crew inputs and real time. The FCOS must guarantee the proper synchronization to maintain identical inputs. For example:

- Sensor and crew inputs must be commanded only after a proper synchronization sequence;
- If all machines possess independent real time clocks, then when real time is desired, the machines must synchronize, exchange real time, and utilize a properly defined average value to be used in navigation and control loop calculations.

To keep the GPCs in synchronization the following functions are employed.

a) Synchronization points are specified. For example, the following synchronization points can be chosen.

- Sync upon entrance to a foreground routine;
- Sync before a data input sequence;
- Sync before a comparison and output cycle;
- Sync upon entrance to a trap routine;
- Sync upon entrance to (or exit from) a background/foreground update block;
- Sync before the real time clock is read and exchanged as data;
- Sync upon entrance to the interrupt service routine; etc.

b) A maximum time-out function is specified. This function represents the maximum waiting time allowed for the machines to synchronize. Different sync points can possess different time-out limits.

c) A topological sync-connection function. This function designates the GPCs with which synchronizationis to occur at the underlying point.

In the Space Shuttle orbiter avionics system a GPC software synchronization technique is thus incorporated into the software system to support simultaneous operation of GPCs in a Redundant Set. It alos supports all active GPCs for System Software Interface Processing.

The following software requirements are associated with the synchronization procedure:

a) The synchronization technique is required to meet time skew constraints, for sampling data sensors and providing output commands to the external voters.

Allowable time skew on inputs is bounded by a specified value denoted as ΔT_I. Typically,

$$\Delta T_I \approx 450 \text{ μsec}$$

Allowable time skew on output commands is bounded by a specified value denoted as ΔT_0. Typically,

$$\Delta T_0 \approx 1 \text{ msec}$$

The input time skew is defined as the time span between the first and last input command to the buses of a redundant sensor set to achieve the effect of a simultaneous read operation. Additional time skews need to be incorporated to account for differences in bus transmission times and sensor response times.

The output time skew is defined as the time difference involved in the issuance of redundant output commands to the buses. Additional time skews need to be incorporated to account for hardware related time differences.

b) The synchronization technique needs to support the fault detection and identification software function. This involves GPC self-test procedures in the simplex mode and additional bit-by-bit comparisons of specified output commands in the redundant mode.

c) The synchronization technique needs to support synchronization of all active GPCs (common sync points) to facilitate system software interface processing.

In particular, SSIP processes are those required to run at the same time in all active GPCs, regardless of the major function they support. For example, the following functions are elements of SSIP processing. (These elements may employ various sync points.)

A. Intercomputer communications (ICC).

B. Time management-required for the input coordination function on the reading of the MTU and passing GPC prime clock values.

C. Downlist control to insure a phase relationship of the downlist program.

D. Configuration change coordination - Required for switch and keyboard inputs that require coordinated configuration changes.

E. Systems status data for display and control - There are numerous parameters in the system software that are required to be available for display across all GPCs. There also are various logic control parameters denoting systems software status required to be passed among all GPCs (for example, what GPCs contain which memory configuration.

F. Applications interfact - Involves the trading of data between dissimilar GPCs to support integrated displays and special interfaces.

G. Launch Data Bus control - Involves changing command configuration of the two LDBs when a request to transfer control is received.

H. GPC initialization - Requires ICC to establish the current configurations of other active GPCs.

I. Annuciation - Common for all memory configurations and required ICC coordination to facilitate GPC control of the PL and DEU buses to output all C&W and alert messages and to combine identical messages produced by a Redundant Set into single messages.

J. GPC error handling - System error responses may require GPC coordination to determine what logic to invoke (for example, to avoid downmoding all GPCs in a redundant set for common mode errors).

K. Mass Memory contention coordination - Involves coordination between GPCs when different configuration require use of a shared Mass Memory Unit.

I.8.2 A Queueing Model

We present a general queueing model to describe message delays and buffer behavior under a synchronization operation.

The unit under consideration need to synchronize a process (being an output or input process) with another process. The other process can be associated with another unit, or be the average process generated from processes associated with a set of network units.

Sync points are determined for the time comparison of the two processes. To model this comparison operation, we assume that underlying messages need to be stored in the unit buffer and queued for a certain time until a time comparison task is completed.

The period of time required for such a message to be queued in the buffer, denoted as S, can be simply represented by the formula

$$S = \frac{\mu^{-1}}{C} + \Delta T_S + \Delta T_D + \Delta T_P , \qquad \text{(I.8.2-1)}$$

where

μ^{-1} = average sync message length [bits];

C = unit processing rate [bits/sec];

ΔT_S = time-skew due to clock differences;

ΔT_D = time-skew due to differences in propagation delays

ΔT_P = time-skew due to hardware processing differences.

If sync points are determined in such a manner that sync messages arrive as a Poisson process at a rate

λ_S = arrival rate of sync messages [bits/sec],

then the unit system under consideration can be considered as a queueing system.

In particular, applying the queue-size and message delay results presented in previous secitons we obtain the following formulas.

The system traffic intensity is given by

$$\rho = \lambda S . \qquad \text{(I.8.2-2)}$$

We require

$$\rho < 1 , \qquad \text{(I.8.2-3)}$$

to ensure finite limiting queue-size and message delay values. Then, the mean buffer queue size $\overline{X}$, representing the average number of messages in the system, queued in the buffer or under processing, is given by

$$\overline{X} = \rho + \frac{1}{2} \frac{\rho^2}{1-\rho} . \qquad \text{(I.8.2-4)}$$

The mean delay (response-time) $\overline{D}$ of a message, representing the amount of time the message has to spend in the buffer for both queueing and processing purposes, is given by

$$\overline{D} = S[1 + \frac{1}{2} \frac{\rho}{1-\rho}] . \qquad \text{(I.8.2-5)}$$

Using these formulas, network constraints upon buffer queue-size and message delays can be applied to deduce the proper constraints upon the underlying time-skew functions.

I.8.3 Clock Synchronization Procedures

We consider the problem of time synchronization for the Shuttle computers, the data bus and other Shuttle systems using the time

functions.

The two main methods that can be applied to synchronize the GPC (or other unit oscillator) can be classified as:

- Master-Slave Sync Techniques
- Mutual Lock Synchronization Techniques

Under the master-slave sync method, one oscillator is named the master and is the frequency reference. The other oscillators are synchronized to the master using phase lock loops. Failure of the master oscillator must be detected whereupon another oscillator is named the master.

Successive master oscillators are selected in order from the surviving oscillators. There are two problems involved in this scheme: Since the entire system operation depends upon proper operation of the master oscillator, failure of the master oscillator must be detected and corrected. Two-failure tolerant failure detection is cumbersome. Also, the circuitry must be reconfigures to select a new oscillator to be the master from the remaining surviving oscillators.

The mutual lock synchronization scheme works as follows. Each oscillator is controlled by a filter, in this case a phase lock loop. The outputs of all four oscillators are added together and applied to the inputs of each phase lock loop. The phase detector at each phase lock loop input determines the relative phase between a particular oscillator and each component of the summed input.

For example, if the oscillator outputs are considered to be sinusoids, the summed outputs will be

$$e_0 = \sum_{i=1}^{4} A_i \sin(\omega c_i t + \phi_i)$$

where the ϕ_i's are measured with respect to some arbitrary but consistent reference. Now the j^{th} phase detector measures the phase difference between the j^{th} oscillator and each of the i components, and it outputs the sum of these phase differences. Thus, the j^{th} detector output is

$$\phi_j = \sum_{i=1}^{4} (\phi_i - \phi_j)$$

where j takes on the values 1,2,3,4.

It can be shown that, as a result of this summing of phase error at each input, the several oscillators will achieve mutual synchronization with normal loop dynamics. This is true provided the center frequencies are within a mutual "pull range" to begin with.

Therein lies the key to failure safe operations for the mutual lock method. The tracking range of each oscillator is limited by clamps of the frequency control input of the oscillator. When an oscillator fails off frequency, loss of lock is assured by properly limiting the pull range. The failed oscillator will then be off frequency and will be properly ignored by the remaining phase lock loops due to the selection of phase lock loop bandwidth smaller than the failed frequency shift. The important point here is that failure of an oscillator does not cause detriment to the remaining oscillators because any oscillator introduces vital control into the loop only when a proper signal is present.

The oscillator used in the clocking circuit can fail in the following ways:

- No output
- Wrong output levels
- Small frequency drift
- Large frequency shift

The first two failure modes can be easily detected by comparison of the performance of the quad computers and will not be detected in the clocking scheme proposed. The second two, however, can cause erroneous calculations of a more subtle nature and must be monitored and any failure rectified.

A detector can be implemented to determine the frequency error between any oscillator and a reference oscillator. The difficulty here is that the reference oscillator may fail or the comparison circuit may fail. The failure modes of the reference oscillator are the smae as for an operational oscillator. The failure detector circuit (comparison circuit) on the other hand may fail in one of two ways: 1) it may erroneously indicate a failure of an oscillator (failure in the FAIL state) or 2) it may erroneously indicate that an oscillator is operational (failure in the GOOD state).

Therefore, it is imperative in the oscillator failure detection scheme to provide that frequency error detection be done without introducing added failure modes. Oscillator frequency error can be determined in two ways. First, it can be deduced by comparing computer calculations using data derived from each reference oscillator. Secondly, oscillator failure can be determined by employing a double-fail-tolerant oscillator failure detector.

In order to use the master-slave synchronization technique, failure detection of the master oscillator must be done followed by an electronic reconfiguration to select a new master oscillator. In order to maximize hardware efficiency, failure detection may be done by a comparison between operational oscillators. Such a comparison between two socillators gives, not a positive indication of failure of either oscillator, but is a failure syndrome indicator; the failure can be either of the oscillators or the failure detector. The failure of a particular oscillator can be determined by taking a majority vote amongst several syndrome indicators, depending upon the number of failures to be tolerated.

In turn, a clock system using a mutual failure detection principle can be used. Such a scheme is designed to guarantee positive failure indication of the five oscillators in spite of any three failures of oscillators or detection circuitry. Oscillator failure is announced any time two syndrome indicators go to the FAIL state. Those syndromes associated with the failed oscillator are then removed from service and no more comparisons accepted from them for additional failure indications. This requires memory of prior failures and also control functions between failure indication (FI) logic. Because of the need to have a three-failure tolerant failure detection scheme, the FI logic must be triple redundant with fail proof wired "OR" failure indication.

The drawbacks in the mutual failure detecting clock system are as follows. The control exerted by one oscillator and its failure circuitry upon the others paves the way for catastrophic failure of one unit to destroy the others Therefore, when oscillator failure

detection is incorporated, the failure detection should be done on a basis wherein independence is maintained between the four clocking subsystems. In general, when a mutual synchronization procedure is employed the structure shown in Fig. I.8.1 can be be employed.

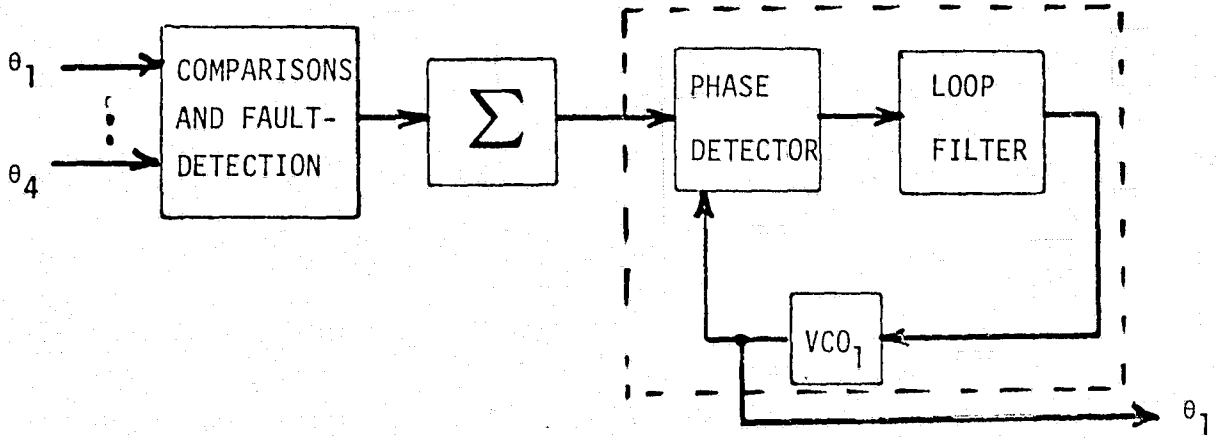

Figure I.8.1.

Comparisons and fault-detection procedures are used upon the received processes (phases), in establishing the integrity of the underlying clocks. Subsequently, the healthy phase processes are summed to yield an average phase process. The latter feeds the phase-locked loop of the system (GPC) under consideration, as shown in Fig. I.8.1 (for GPC number 1).

The system analysis of such a loop is carried out in the following manner. Consider the PLL model for oscillator number 1 shown in Fig. I.8.2.

Neglecting the VCO tuning voltage and VCO instability one can write the stochastic nonlinear differential equation by inspection as follows:

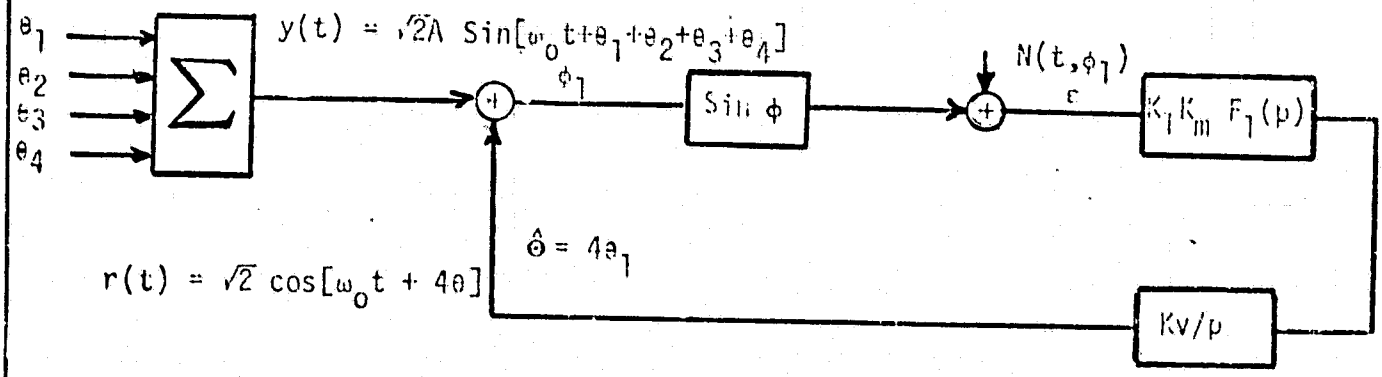

Figure I.8.2

$$\hat{\theta} = \frac{Kv}{p}\,[F_1(p)\,\varepsilon]$$

$$= \frac{K\,F_1(p)}{p}\,[\mathrm{Sin}\,(\phi_1) + N(t,\phi_1)] \qquad \text{(I.8.3-1)}$$

where

$\phi_1 = (\theta_1 + \theta_2 + \theta_3 + \theta_4 - 4\theta_1)$ = phase error

$N(t,\phi_1) \triangleq N_c \cos\phi_1 - N_s \sin\phi_1$ = equivalent phase noise

We can write

$$\phi_1 = \sum_{i=1}^{4} \theta_i - \hat{\theta} = (\theta_1 + \theta_2 + \theta_3 + \theta_4) - \frac{K\,F_1(p)}{p}\,[\mathrm{Sin}\,\phi_1 + N(t,\phi_1)]$$

$$\phi_1 = \sum_{i=1}^{4} \theta_i - \frac{K\,F_1(p)}{p}\left[\sum_{i=1}^{4} \mathrm{Sin}(\theta_i - 4\theta_1) + N(t,\phi_1)\right] \qquad \text{(I.8.3-2)}$$

where,

$$\phi_1 = (\theta_1 + \theta_2 + \theta_3 + \theta_4 - 4\theta_1)$$

Similarly, the equations for other three loops may be written

as follows:

$$\phi_2 = \sum_{i=1}^{4} \theta_i - \frac{K F_2(p)}{p}\left[\sum_{i=1}^{4} \text{Sin}\ (\theta_i - 4\theta_2) + N(t, \phi_2)\right] \quad \text{(I.8.3-3)}$$

where

$$\phi_2 = (\theta_1 + \theta_2 + \theta_3 + \theta_4 - 4\theta_2)$$

$$\phi_3 = \sum_{i=1}^{4} \theta_i - \frac{K F_3(p)}{p}\left[\sum_{i=1}^{4} \text{Sin}(\theta_i - 4\theta_3) + N(t, \phi_3)\right] \quad \text{(I.8.3-4)}$$

where

$$\phi_3 = (\theta_1 + \theta_2 + \theta_3 + \theta_4 - 4\theta_3)$$

$$\phi_4 = \sum_{i=1}^{4} \theta_i - \frac{K F_4(p)}{p}\left[\sum_{i=1}^{4} \text{Sin}(\theta_i - 4\theta_4) + N(t, \phi_4)\right] \quad \text{(I.8.3-5)}$$

where

$$\phi_4 = (\theta_1 + \theta_2 + \theta_3 + \theta_4 - 4\theta_4).$$

Equations (I.8.3-2) to (I.8.3-5) represent the system equations for four parallel coupled loops. Each equation is a nonlinear stochastic differential equation with coupling introduced due to other phase lock loops. By assuming $F_1(p) = F_2(p) = F_3(p) = F_4(p) = 1$, i.e., a first order loop and linearizing so that $\sin \phi \approx \phi$, the Fokker Planck technique of analysis can be applied to solve the simplified equations.

To illustrate the efffects of time delays between oscillators, consider the model shown in Fig. I.8.3.

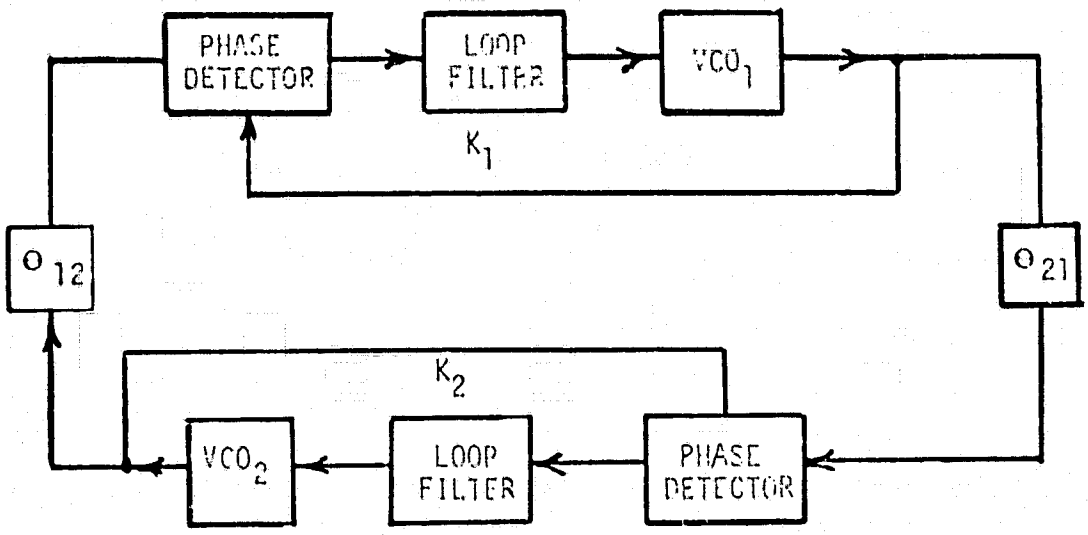

Figure I.8.3.

The fundamental equation of a single phase-locked loop is given by

$$W_1 = W_{01} + K_1 \cos (\phi_2 - \theta_{12} - \phi_1)$$

where,

W_1 = Synchronized output signal frequency

W_{01} = Nominal frequency of the controlled oscillator

K_1 = loop gain of the controlled oscillator in rad/sec/rad.

ϕ_1 and ϕ_2 = relative output phases of oscillators one and two

θ_{12} = phase delay from oscillator 2 to oscillator 1

W_{ss} = steady state network frequency

For the two phase locked loop clocks the steady state equations are:

$$W_1 = W_{01} + K_1 \cos(\phi_2 - \theta_{12} - \phi_1) \qquad \text{(I.8.3-6)}$$

$$W_2 = W_{02} + K_2 \cos(\phi_1 - \theta_{21} - \phi_2) \qquad \text{(I.8.3-7)}$$

where $W_{ss} = W_1 = W_2$ = the steady state output frequency of both oscillators.

For a practical network let

$$W_{01} = W_{02} \quad \text{and} \quad K_1 = K_2 = K.$$

Then equation (I.8.3-6) and (I.8.3-7) gives

$$0 = W_{01} - W_{02} + K[\cos(\phi_2 - \phi_1 - \theta_{12}) - \cos(\phi_1 - \phi_2 - \theta_{21})]$$

$$-\frac{W_{01} - W_{02}}{K} = \sin\left[\frac{-(\theta_{12} + \theta_{21})}{2}\right] \sin\left[\frac{2\phi_2 - 2\phi_1 + (\theta_{21} - \theta_{12})}{2}\right]$$

Let $\sin\left[\frac{-(\theta_{12} + \theta_{21})}{2}\right] = A$

$$\phi_2 - \phi_1 = \phi$$

and $$\frac{\theta_{21} - \theta_{12}}{2} = 0$$

Substitute these in the above equation to give

$$\frac{\Delta W_o}{2AK} = \sin(\phi + \theta)$$

$$\phi + 0 = \sin^{-1}\frac{\Delta W_o}{2AK}$$

For a practical case

$$\Delta W_o \simeq 0 \quad \text{and } K >> 1.$$

$$\phi = -\theta$$

Substitute this in equation (I.8.3-6) to get

$$W_1 = W_{01} + K \cos \frac{(-\theta_{21} + \theta_{12} - 2\theta_{12})}{2}$$

Thus in general

$$W_{ss} = W_1 = W_{01} + K \cos \left[\left(\frac{\theta_{12} + \theta_{21}}{2}\right)\right] \qquad \text{(I.8.3-8)}$$

These methods can now be integrated with the queueing techniques presented previously in this section and the reliability methods developed in the following sections, to obtain global system performance characteristics.

II. SYSTEM RELIABILITY MEASURES AND COMMUNICATION PATH FAILURE ANALYSIS

II.1 Reliability Features of the Data Processing Network

The Space Shuttle orbiter avionics system is described in Section I.1. In this section we will summarize the main system reliability features.

The Space Shuttle avionics system contains five general purpose computer (GPCs) communicating with the avionic subsystem over a network of serial data buses (see Figs. I.1.1-I.1.2). Four of the five GPcs are identically programmed to perform flight-critical functions, such as guidance, navigation and control. The fifth computer is programmed to perform non-flight-critical avionic functions.

Subsystems that perform similar functions are assigned to the same data-bus group. There are seven such groups (Fig. I.1.1). The subsystems have varying levels of redundancy at the unit level, depending on their criticality. To prevent the loss of more than one redundant unit when one data bus fails, no two redundant units interface with the same bus.

During time-critical mission phases (when recovery time is less than one second), such as boost, reentry and landing, four of the five GPCs operate as a redundant set, receiving the same input data, performing the same flight critical computations and transmitting the same output commands. In this mode of operation, efficient detection and identification of two flight critical computer failures is provided by comparing the output commands and "voting" on the results. This is called the voting subsystem.

After two failures, the remaining two computers in the set use comparison and self-test techniques to provide tolerance of a third failure. The voting mechanism thus allows a computer to transmit incorrect commands to critical subsystems for an indefinite number of cycles without having adverse effects on system operation.

Each of the redundant subsystems is connected to a different bus. Thus, a different computer requests data from each of the subsystems and the returned data are available to all other computers in the set.

In non-critical phases of the mission, each of the GPCs is associated with a proper dedicated subset of subsystems. This non-redundant configuration is termed the simplex mode.

Topologically, we note that the data processing system is structuared around a central set of GPCs. Thelatter are interconnected to the subsystems so that they can be operated in redundant groups to provide critical services.

Interface adaptation between the data bus network and the orbiter subsystems is accomplished by multiplexer/demultiplexer (MDM) units. The GPC complex is interfaced with the data bus network through the set of I/O processors (IOPs). The serial digital data buses are time-shared, so that data transfer is carried on a time-division multiplexed (TDM) basis, using pulse code modulation (PCM).

Each GPC contains a self-testing program as well as built-in test equipment. The latter enables it to attain a 96% fault detection capability.

Each computer IOP interfaces with the other IOPs and with the interfacing subsystems over the 24 separate serial data buses. The IOP contains a set of 24 independent processors, called Bus Control Elements (BCEs). A 25th processor, the Master Sequence Controller (MSC) controls the operation of the 24 BCEs. These 25 processors act as separate digital computers, with data processing programs independent of the CPU programs. Each BCE controls a Multiplexer Interface Adapter (MIA), which is connected to the serial data buses via bus couplers (see Fig. I.1.3). The MIA transmits and receives information, encodes and decodes bus data, and tests for parity and proper synchronization of bits.

In describing the reliability, fault detection and failure properties of the avionics data processing network, we will identify the relevant failure and reliability measures and models for: the computer system; the communication network; the subsystem complex; and the proper integrated interfaces among these subnetworks.

II.2 Failure Parameters and Reliability Performance Measures for the Computer Complex

In considering failures of system elements, we examine failures associated with the computer system, the communication network and the application subsystems.

We first consider failures associated with the computer complex. A GPC is assumed to have an average failure rate equal to λ_c [failures/sec], so that

$$\lambda_c = \text{average GPC failure rate [failures/sec]} \qquad \text{(II.2-1)}$$

The period of time from initiation of operation to the failure of a GPC, is called the PGC lifetime. It is a random variable, denoted as T_c. Thus

$$T_c = \text{GPC lifetime} = \text{GPC operational time til failure [sec]} \qquad \text{(II.2-2)}$$

The mean duration $E(T_c) = \overline{T}_c$ is equal to λ_c^{-1},

$$E(T_c) = \overline{T}_c = 1/\lambda_c \ . \qquad \text{(II.2-3)}$$

To statistically characterize T_c we need to specify its distribution function $F_c(x)$,

$$F_c(x) = P(T_c \leq x) \ , \ x > 0 \ . \qquad \text{(II.2-4)}$$

It is many times assumed that T_c is exponentially distributed, so that

$$F_c(x) = 1 - e^{-\lambda_c x} \ , \ x > 0. \qquad \text{(II.2-5)}$$

Other lifetime distributions are sometimes also used. For example, a useful two parameter lifetime distribution is the Gamma distribution with parameters $\lambda > 0$ and $k = 1,2,\ldots$, given by the density

$$f_c(x) = \frac{d}{dx} F_c(x) = \frac{\lambda}{(k-1)!} (\lambda x)^{k-1} e^{-\lambda t}, \ t > 0. \qquad \text{(II.2-6)}$$

Another useful lifetime distribution is the Weibull distribution with parameters v and k, $v \geq \varepsilon$, $k > 1$, given as

$$F_c(x) = \begin{cases} 1 - \exp\left\{ - \left(\frac{x-\varepsilon}{v-\varepsilon} \right)^k \right\}, & x \geq \varepsilon \\ 0 \ , & x < \varepsilon \end{cases} \qquad \text{(II.2-7)}$$

The conditional failure rate function $h_c(x)$, also called the hazard function, is given by

$$h_c(x) = \frac{f_c(x)}{1-F_c(x)} \quad . \qquad \text{(II.2-8)}$$

The hazard function $h_c(x)$ yields the density of computer failure after a lifetime of duration x, given that it has not failed during its first x units of time of operation. Thus:

$$h_c(x)dx = P\{x < T \leq x + dx | T > x\} \quad . \qquad \text{(II.2-9)}$$

For a Weibull lifetime distribution, with parameters , v and k, we have

$$h_c(t) = k\left(\frac{t-\varepsilon}{v-\varepsilon}\right)^{k-1} \quad . \qquad \text{(II.2-10)}$$

Thus, the chance of a GPC failure increases with time in accordance with expression (II.2-10).

For an exponential lifetime distribution (II.2-5) with parameter λ_c, we obtain

$$h_c(x) = \lambda_c \text{ , for each } x > 0 \; . \qquad \text{(II.2-11)}$$

Thus, under an exponential lifetime distribution, the conditional GPC failure rate is constant. The chance that a GPC will currently fail, given it has not yet failed, is independent of the length of time this GPC has been operational. The exponential distribution is therefore memoryless. A non-exponential distribution, such as the Gamma or Weibull distribution, should be used if the GPC conditional failure rate cannto be assumed to be constant. In general, the GPC conditional failure rate is a non-decreasing function of the past GPC lifetime.

Considering a simplex operation of a GPC, self-test tests and programs are used to detect a computer failure. The probability of a computer failure detection, using only self-test techniques is called the computer coverage. Thus, we set

$$P_d = \text{GPC coverage probability}$$

$$= P\{\text{failure detected by GPC self-test operation} \mid \text{GPC failure occurred}\} \quad . \qquad \text{(II.2-12)}$$

In the Space Shuttle avionics system, a goal of 96% coverage of computer failures has been set, when no external test equipment or cooperative use of other GPCs is employed.

To obtain

$$P_d = 0.96 \quad ,$$

all GPC self-test techniques are employed, including: built-in test equipment, timer micro and macro-coded self testing procedures. A storage of CPU 110 half-words and a CPU processing time of 1.3 msec is required.

To attain a coverage of

$$P_d = 0.88 \quad ,$$

the above mentioned macro-coded self-testing procedure can be withdrawn. Then, a CPU storage of only 14 half-words and a CPU processing time of only 0.15 msec is required.

It is worthwhile to achieve $P_d = 0.96$ prior to assigning a GPC to a redundant set. However, to save storage and processing time in using self-test procedures in the redundant set, during critical mission phases, it is sufficient to attain $P_d = 0.88$.

The resulting redundant set reliability measure will be evaluated in a later section.

The build-in test equipment by itself can yield $P_d = 0.37$. It requires virtually no additional CPU storage and processing resources.

It is also of interest for certain mission purposes, to model secondary GPC failures. These are failures that do not affect the operation of the GPC as related to the present mission. Given that a GPC failure has occurred, we let P_{SF} be the probability that it is a secondary failure. Thus:

$$P_{SF} = P\{\text{failure is secondary} \mid \text{GPC failure has occurred}\} . \quad \text{(II.2-13)}$$

Thus, we have

$$\lambda_{CS} = \text{GPC secondary failure rate} = P_{SF}\lambda_c , \quad \text{(II.2-14a)}$$

$$\lambda_{CP} = \text{GPC primary failure rate} = (1-P_{SF})\lambda_c . \quad \text{(II.2-14b)}$$

It is also possible to differentiate between transient and permanent GPC failures. A transient GPC failure will cause an incorrect computer output which can be restored within a relatively short period of time T_{TR}. A much longer restoration time T_{PR} is required to correct a permanent failure. The corresponding mean restoration times are

$$\overline{T}_{TR} = E[T_{TR}] , \quad \text{(II.2-15a)}$$

$$\overline{T}_{PR} = E[T_{PR}] . \quad \text{(II.2-15b)}$$

Restoration times are sometimes assumed to be exponentially distributed, but any proper distribution (such as a Gamma distribution) can be assumed. For critical mission phases, we can set $\overline{T}_{PR} = +\infty$.

In detecting computer failures, use is made of mutual tests and data interchange between GPCs, of inter-GPC comparisons, as well as of self-test procedures. We set

$P_d(N)$ = probability of detecting a GPC failure, given it has occurred, when both self-test procedures and comparison procedurs among N GPCs are used. (II.2-16)

Clearly, we have

$$P_d = P_d(1) \quad , \tag{II.2-17}$$

and

$$P_d(N) \geq P_d(N-1) \quad , \quad P_d(N) \geq P_d, \; N \geq 1. \tag{II.2-18}$$

In choosing reliability performance measures to assess the failure invulnerability of the Space Shuttle avionics computer complex, we consider the two computer system modes: the simplex mode and the redundant mode.

In the simplex mode, an operating GPC serves a certain set of subsystems. To assess its operational reliability we define the following indices.

$Q_{CF}(T)$ = Probability of a computer failure within T sec of operation, in simplex mode. (II.2-19)

$\bar{L}_{CF}$ = mean time between GPC failures (MTBF), in simplex mode. (II.2-20)

If we incorporate computer restoration operations, then we are also interested in the following performance function:

Q_{CO} = probability of a GPC being in a failure state, under restoration, in simplex mode (II.2-21)

In the simplex mode, when a computer fails, it can be replaced by another one. It is assumed that a minimum of two GPCs is required for regular operation. One is then interested in computing the simplex system loss probability:

$Q_{SL}(T)$ = the simplex system loss probability

= probability that there are no two operational GPCs, in simplex mode, in T units of time. (II.2-22)

We turn now to consider the redundant computer system mode. In this mode, 4 GPCs are operating in parallel, performing identical information processing operations. Comparisons are made between the computer outcomes. A voting procedure is then employed. The failure of one or two GPCs is immediately identified and GPC-located by the voting mechanism. The failure of a third GPC is indicated by the voting procedure. However to detect which of the remaining two GPCs has failed, self-testing procedures are utilized.

We assess the computer-complex reliability performance in the redundant mode by the following measures.

$P_{SL}(T)$ = probability of a computer system loss during a T sec redundant computer system operation. (II.2-23)

The redundant computer system is said to be lost, during a mission phase of T sec duration, if no GPC is remained operational.

In assessing the increase in reliability contributed by the number of redundant parallel GPCs (denoted as N), we are also interested in computing the index:

$P_{SL}(T,N)$ = probability of system loss during a T sec redundant operation of N GPCs (II.2-24)

We note that in the present system, N = 4, so that

$$P_{SL}(T) = P_{SL}(T,4) . \qquad \text{(II.2-25)}$$

The following mean time between failures also provides a measure of redundant system invulnerability.

$\overline{T}_F(N)$ = mean time to system failure of a redundant N-GPC computer complex . (II.2-26)

In the present system N=4, so that we set

$$\overline{T}_F = \overline{T}_F(4) . \qquad \text{(II.2-27)}$$

II.3 Failure Analysis for the Computer System: The Simplex Mode

II.3.1 Single GPC Failure Analysis

In the simplex mode, each of the GPCs is associated with a proper dedicated subset of subsystems.

Assume that out of the N available GPCs, only M GPCs are used on a dedicated basis, $M \leq N$. The remaining N-M GPCs are used to replace failing GPCs.

Each GPC is governed by a failure rate λ_c. (Assume only primary failures.) Using self-testing procedures, the probability of detecting a GPC failure, once it has failed, is equal to P_d.

The mean time to failure of a GPC is thus equal to

$\overline{L}_{CF} = \overline{T}_F(1)$ = mean time to failure for a simplex GPC

$$= 1/\lambda_c . \qquad \text{(II.3.1-1)}$$

If the time to failure L_{CF} of a single GPC is exponentially distributed we have

$$P(L_{CF} > t) = e^{-\lambda_c t} , \quad t > 0 . \qquad \text{(II.3.1-2)}$$

A time-dependent self-testing failure detection process is described as follows. We set

L_{FD} = time to failure detection, by self-testing techniques, for a simplex GPC, given failure has occurred. (II.3.1-3)

The mean time to failure detection $\bar{L}_{FD}$ is equal to

$$\bar{L}_{FD} = E[L_{FD}] = \lambda_d^{-1} , \quad \text{(II.3.1-4)}$$

provided failure detection occurs. If L_{FD} is exponentially distributed, we set

$$P(L_{FD} > t) = (1-P_d) + P_d e^{-\lambda_d t}, \quad t > 0. \quad \text{(II.3.1-5)}$$

Therefore, we conclude that

Probability of a GPC undetected failure in t units of time

$$= \int_0^t P[L_{CF} \epsilon (u,u+du)] P(L_{FD} > t-u)$$

$$= \int_0^t \lambda_c e^{-\lambda_c u} [P_d e^{-d(t-u)} + 1-P_d] du$$

$$= \frac{\lambda_c P_d}{\lambda_c - \lambda_d} [e^{-\lambda_d t} - e^{-\lambda_c t}] + (1-P_d)(1-e^{-\lambda_c t}) . \quad \text{(II.3.1-6)}$$

Thus, with probability $1-e^{-\lambda_c t}$ a GPC will fail within t units of time. After failure, by self-testing techniques its failure will be detected with probability P_d, and undetected probability $1-P_d$. The dynamics of failure detection is described by Eq. (II.3.1-5). The latter yields the probability that failure detection (by self-testing) will require more than t units of time. Eq. (II.3.1-6) describes the probability that a GPC failure will occur within t units of time

and that the failure will remain undetected during this period.

II.3.2 Failure Analysis for the Simplex Computer System

We consider the computer complex under the simplex mode. It is assumed that M GPCs need to be used on a regular basis, each being assigned a dedicated set of subsystems. The total number of available GPCs is equal to N, N > M. For the avionics system, we typically have N = 5, M = 2. The failure characteristics of each GPC haye been analyzed in the previous section.

We assume now that upon the detectionof a failed GPC, it is immediately replaced by an in reserve GPC, if such is available. Initially, M GPCs are operating and N-M GPCs serve as reserve units. We say the system loss has occurred when no more than M-1 operating GPCs are left. Thus, we set

$$Q_{SL}(T,M) = P\{\text{no more than M-1 GPCs are left}\} . \qquad \text{(II.3.2-1)}$$

For the avionics system, M=2, so that

$$Q_{SL}(T) = Q_{SL}(T,2) . \qquad \text{(II.3.2-2)}$$

We wish to compute $Q_{SL}(T,M)$ and $Q_{SL}(T)$.

The GPC failure point process can be noted to be a Poisson process with rate $M\lambda_c$ [failures/sec]. We subsequently obtain the following result.

$$Q_{SL}(T,M) = P\{\text{more than N-M+1 computer failures in T units of time}\}$$

$$= \int_0^T \frac{M\lambda_c}{(N-M)!} (M\lambda_c u)^{(N-M)} e^{-M\lambda_c u} du$$

$$= 1 - \sum_{n=0}^{N-M} e^{-M\lambda_c T} \frac{(M\lambda_c T)^n}{n!} . \qquad \text{(II.3.2-3)}$$

Therefore, for N=5,M=2, we obtain:

$$Q_{SL}(T) = 1 - \sum_{n=0}^{3} e^{-2\lambda_c T} \frac{(2\lambda_c T)^n}{n!}$$

$$= 1 - e^{-2\lambda_c T}\left\{1 + 2\lambda_c T + \frac{(2\lambda_c T)^2}{2!} + \frac{(2\lambda_c T)^3}{3!}\right\} . \qquad \text{(II.3.2-4)}$$

Using expression (II.3.2-4), we can thus compute the probability $Q_{SL}(T)$ that the simplex system fails, so that no more than a single GPC is operating in T units of operation time. Alternatively, given a desired maximum simplex loss probability q_0, we can evaluate the critical time T_c such that

$$T_c = \max\{T: Q_{SL}(T) \leq Q_0\} . \qquad \text{(II.3.2-5)}$$

To compute T_c, we solve

$$Q_{SL}(T_c) = Q_0 . \qquad \text{(II.3.2-6)}$$

II.3.3 Restoration Analysis for the Simplex Computer System

We consider the simple computer system presented in the previous section, but now assume that failed GPCs can be restoaed. We assume the GPC restoration time T_R to be exponentially distributed

$$P(T_R > t) = e^{-\lambda_R t} , \quad t > 0, \qquad \text{(II.3.3-1)}$$

with a mean restoration time $\overline{T}_R$ equal to

$$\overline{T}_R = E[T_R] = \lambda_R^{-1} . \qquad \text{(II.3.3-2)}$$

Computer time to failure is exponentially distributed with mean λ_c^{-1}. Assume here that $P_d = 1$. There are altogether N GPCs. Only M GPCs can be used simultaneously where $M < N$.

To analyze the statistical characteristics of this computer system, we model it as a proper queueing network, shown in Fig. II.3.3.1.

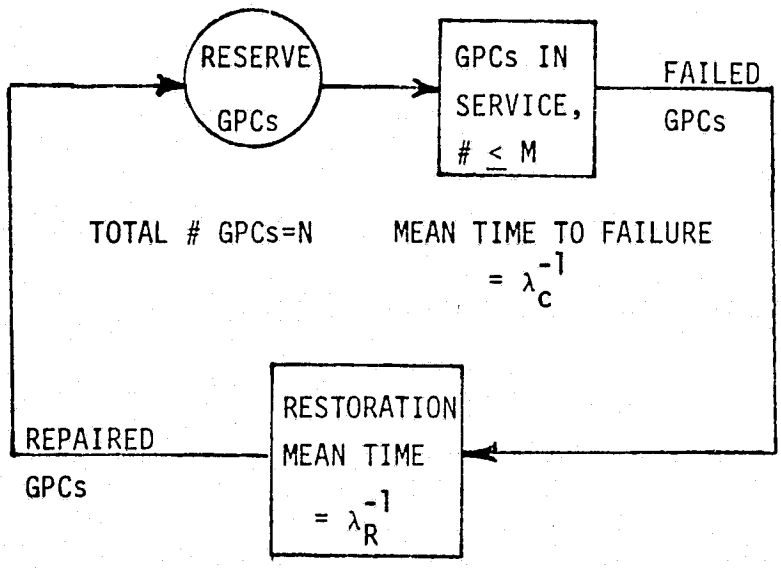

Figure II.3.3.1

In this queueing network, no more than M GPCs can be used in parallel. Each will fail after an average operating time equal to λ_c^{-1}. Upon its failure, a GPC is being restored. Average restoration time is equal to λ_R^{-1}. When a GPC is restored, it immediately joins the queue of reserve GPCs. Whenever the number of GPCs in service becomes below M, a reserve GPC (if available) enters service.

To analyze this system, we use and extend the methods developed in Sections I.6.5-I.6.6. We set

$$P_n = P\{\text{n GPCs are in the system, operating condition, in reserve or being used}\} . \qquad \text{(II.3.3-3)}$$

We then obtain the following formulas

$$P_0 = \left\{ \sum_{k=0}^{M-1} \binom{N}{k} M^k \rho^k + \sum_{k=M}^{N} \frac{N!\rho^k}{(N-k)!} \frac{M^M}{M!} \right\}^{-1} , \qquad \text{(II.3.3-4)}$$

where

$$\rho = \frac{\lambda_R}{M\lambda_C} \quad ; \tag{II.3.3-5}$$

and

$$P_k = \begin{cases} P_0 \binom{N}{k} M^k \rho^k, & \text{if } k \leq M-1 \\ P_0 \dfrac{N!\rho^k}{(N-k)!} \dfrac{M^M}{M!}, & \text{if } M \leq k \leq N \\ 0, & \text{otherwise} \end{cases} \tag{II.3.3-6}$$

We can now set the probability of system loss Q_{SL} for this model to be equal to the probability that the system contains no more than M-1 GPCs in working condition. We then obtain Q_{SL} to be given by

$$Q_{SL} = \sum_{k=0}^{M-1} P_k \quad , \tag{II.3.3-7}$$

where P_k is expressed by Eqs. (II.3.3-4)-(I.3.3-6).

In this manner, the system engineer can compute the probability of computer system loss under a simplex mode of operation. The proper system parameters (such as GPC failure rates, restoration rates, number of reserve GPCs) can then be adjusted or chosen.

II.4 Failure Analysis for the Computer System: The Redundant Mode

We compute in this section the underlying reliability performance measures for the computer system under the redundant mode. In this mode, four GPCs are operating in parallel conducting identical operations. The outputs of these GPCs are compared and voting is used to decide upon the correct output. In this manner, one and two GPC failures are readily detected and the failed computer is identified. When only two operating GPCs are left, by comparing outputs one can detect the failure of a third computer. It, however, remains to identify the third failing GPC. Self-testing procedures are subsequently used. When only one GPC is left, only self-testing techniques can be used to detect its failure. The underlying reliability characteristics are then identical with those computed for the simplex mode in Section II.3.

To understand the performance dependence upon the number of parallel GPCs in the redundant mode, we assume that there are N parallel GPCs. In the avionics system under consideration, a number of N=4 parallel GPCs are employed. Thus, we set

$$N = \text{number of parallel GPCs in redundant mode.} \qquad \text{(II.4-1)}$$

Each GPC, using self-testing procedures and programs has a coverage probability P_d. Thus, given a GPC has failed, it will detect its failure with probability P_d, employing self-test techniques.

The GPC failure rate is equal to

$$\text{GPC failure rate} = \lambda_c . \qquad \text{(II.4-2)}$$

We assume only primary failure here. Each GPC has a life-time (time to failure) described by a random variable T_c (see Section II.2). Note that

$$\overline{T}_c = E(T_c) = 1/\lambda_c \quad . \tag{II.4-3}$$

We initially assume that T_c is exponentially distributed (Eq.(II.2-5)).

Typical values for the avionics system are:

$$P_d = 0.96 \; ; \quad \lambda_c = 8 \times 10^{-2} \text{ [failures/hour]} \; . \tag{II.4-4}$$

Our analysis is general, so that any proper parameter values can be incorporated.

We first consider the probability measure $P_d(N)$. It has been defined by Eq. (II.2-16) as the probability of detecting a GPC failure, given it has occurred, when both self-test procedures and comparison procedures among N GPCs are used. In the redundant mode, we employ the comparison-voting procedure to detect and identify failed computers. Therefore, if i GPCs are operating in parallel with $i \geq 3$, we can always perfectly detect and identify any single GPC failures; so that

$$P_d(i) = 1, \text{ if } i = 3,4,\ldots, N \tag{II.4-5}$$

When only two GPCs are operating, we can still perfectly detect whether one of the GPCs has failed, so that

$$P_d(2) = 1 \tag{II.4-6}$$

In this case, however, we need to employ self-test techniques to identify the failed computer.

When a single operating GPC is left, only self-test techniques are used to identify its failures. Subsequently, we have

$$P_d(1) = P_d \ , \qquad \text{(II.4-7)}$$

so that the failure detection probability is equal to the GPC coverage.

To assess the reliability of the redundant computer complex, we are interested in computing the following two measures:

$P_{SL}(T,N)$ = the probability of system loss during a T sec redundant operation, starting with N parallel GPCs; (II.4-8)

$\overline{T}_F(N)$ = mean time to system failure for a redundant computer system, starting with N parallel GPCs. (II.4-9)

The function $P_{SL}(T,N)$ is computed as follows. We set

$$f_{N2}(t)dt = P\{(N-2)\text{nd GPC failure occurs in } (t,t+dt)\}. \qquad \text{(II.4-10)}$$

Thus, $f_{N2}(t)dt$ expresses the probability that, starting with N parallel GPCs, we are left at time t with only two operating GPCs, and the last failure occurred at time t, within (t-dt,t].

If every computer has an exponentially distributed lifetime, with mean λ_c^{-1}, and GPC lifetimes are statistically independent (as well as identically distributed), we obtain the following result.

$$\begin{aligned} f_{N2}(t)dt &= P\{(N-3) \text{ failures in } (0,t)\}P\{\text{a failure in } (f_1 t+dt)\} \\ &= \binom{N}{N-3}(1-e^{-\lambda_c t})^{N-3}(e^{-\lambda_c t})^3 \, 3\lambda_c e^{-3\lambda_c t} dt \ . \end{aligned} \qquad \text{(II.4-11)}$$

Therefore,

$$f_{N2}(t) = \frac{1}{2} \frac{N!}{(N-3)!} \lambda_c (1-e^{-\lambda_c t})^{N-3} e^{-3\lambda_c t} \ . \qquad \text{(II.4-12)}$$

We also note that the times between the first N-2 failures are statistically described as follows. They are i.i.d. random variables such that the time between the i-th and (i+1)st GPC failures is

exponentially distributed with mean $[(N-i)\lambda_c]^{-1}$, for $i = 0,1,\ldots,N-3$.

We now observe the failure of the redundant computer system to proceed in two phases. In the first phase, starting with N parallel GPCs, N-2 GPCs fail. Using the comparison voting procedure, these failures are immediately perfectly detected and identified. We set

$$T_F(N,1) = \text{time duration of first failure phase} = \text{time until the (N-2)nd GPC failure .} \qquad \text{(II.4-13)}$$

Then, $T_F(N,1)$ is governed by the Gamma density (II.4-12). In particular, the probability that phase one will be longer than t sec is given by

$$P\{T_F(N,1) > t\} = \int_t^\infty \frac{1}{2}\lambda_c \frac{N!}{(N-3)!}(1-e^{-\lambda_c x})^{N-3}e^{-3\lambda_c x}dx \ . \qquad \text{(II.4-14)}$$

The mean duration of a phase one mode is given by

$$\overline{T}_F(N,1) = E[T_F(N,1)] = \lambda_c^{-1}\sum_{i=3}^{N} i^{-1} \ . \qquad \text{(II.4-15)}$$

In particular, for N=4 we have:

$$f_{42}(t) = 12\lambda_c(1-e^{-\lambda_c t})e^{-3\lambda_c t} \ ; \qquad \text{(II.4-16)}$$

$$\overline{T}_F(4,1) = \frac{7}{12}\lambda_c^{-1} \ . \qquad \text{(II.4-17)}$$

Upon the termination of phase one, when we are left with only two operating GPCs, the phase-two failure mode starts (provided at this time, the computer system still operates in the redundant mode). Having now two operating GPCs, we are interested in computing the system loss probability $P_{SL}(t,2)$. This is the probability that, starting with 2 PGCs, no operating GPC is left within t units of time. To derive this function, we write:

$$1 - P_{SL}(t,2) = P\{\text{no GPC failures in } (0,t)\} + P\{\text{a single GPC fails in } (0,t)\}P\{\text{detecting a failure} \mid \text{a GPC failure has occurred}\}$$
$$= e^{-2\lambda_c t} + 2P_d e^{-\lambda_c t}(1-e^{-\lambda_c t}) . \qquad \text{(II.4-18)}$$

Therefore, we conclude that

$$P_{SL}(t,2) = 1 - 2P_d e^{-\lambda_c t} - e^{-2\lambda_c t}(1-2P_d) . \qquad \text{(II.4-19)}$$

We can now compute the system loss probability $P_{SL}(T,N)$ as

$$P_{SL}(T,N) = \int_0^T f_{N2}(t) P_{SL}(T-t,2)dt . \qquad \text{(II.4-20)}$$

Substituting (II.4-12) and (II.4-19) into (II.4-20) we obtain the following result:

$$P_{SL}(T,N) = \int_0^T \frac{1}{2}\lambda_c \frac{N!}{(N-3)!}(1-e^{-\lambda_c t})^{N-3} e^{-3\lambda_c t} [1-2P_d e^{-\lambda_c (T-t)} - (1-2P_d)e^{-2\lambda_c (T-t)}]dt . \qquad \text{(II.4-21)}$$

In particular, for the present avionics system we set N=4 in (II.4-21) and obtain, after some algebra, the following expression for the computer system loss probability.

$$P_{SL}(T) = P_{SL}(T,4) = 1 - e^{-2\lambda_c T}(3e^{-2\lambda_c T} - 8e^{-\lambda_c T} + 6) - 4P_d e^{-\lambda_c T}(1-e^{-\lambda_c T})^3$$
$$= (1-e^{-\lambda_c T})^3[1+e^{-\lambda_c T}(3-4P_d)] . \qquad \text{(II.4-22)}$$

Eq. (II.4-22) can also be derived simply as follows. We note that

$$P_{SL}(T) = P\{4 \text{ GPCs fail in } (0,T)\} + P\{3 \text{ GPCs fail in } (0,T)\}(1-P_d)$$
$$= (1-e^{-\lambda_c T})^4 + (1-P_d)4e^{-\lambda_c T}(1-e^{-\lambda_c T})^3 . \qquad \text{(II.4-23)}$$

Eqs. (II.4-22) and (II.4-23) are identical.

Extending the approach used to derive (II.4-23), we obtain the loss probability $P_{SL}(T,N)$ when starting with N parallel GPCs, $N \geq 3$, as follows.

$$P_{SL}(T,N) = P\{N \text{ GPCs fail in } (0,T)\} + P\{N-1 \text{ GPCs fail in } (0,T)\}(1-P_d)$$
$$= (1-e^{-\lambda_c T})^N + (1-P_d)Ne^{-\lambda_c T}(1-e^{-\lambda_c T})^{N-1}$$
$$= (1-e^{-\lambda_c T})^{N-1}[1+e^{-\lambda_c T}(N-1-NP_d)] . \qquad \text{(II.4-24)}$$

Eqs. (II.4-22)-(II.4-24) can now be used to compute the loss probability associated with the redundant computer complex. We note the following characteristics of $P_{SL}(T)$. (Similar properties hold for $P_{SL}(T,N)$, using (II.4-24).)

The loss probability $P_{SL}(T)$, given by (II.4-22), is a linearly decreasing function of the coverage P_d. This is illustrated by Fig. II.4.1.

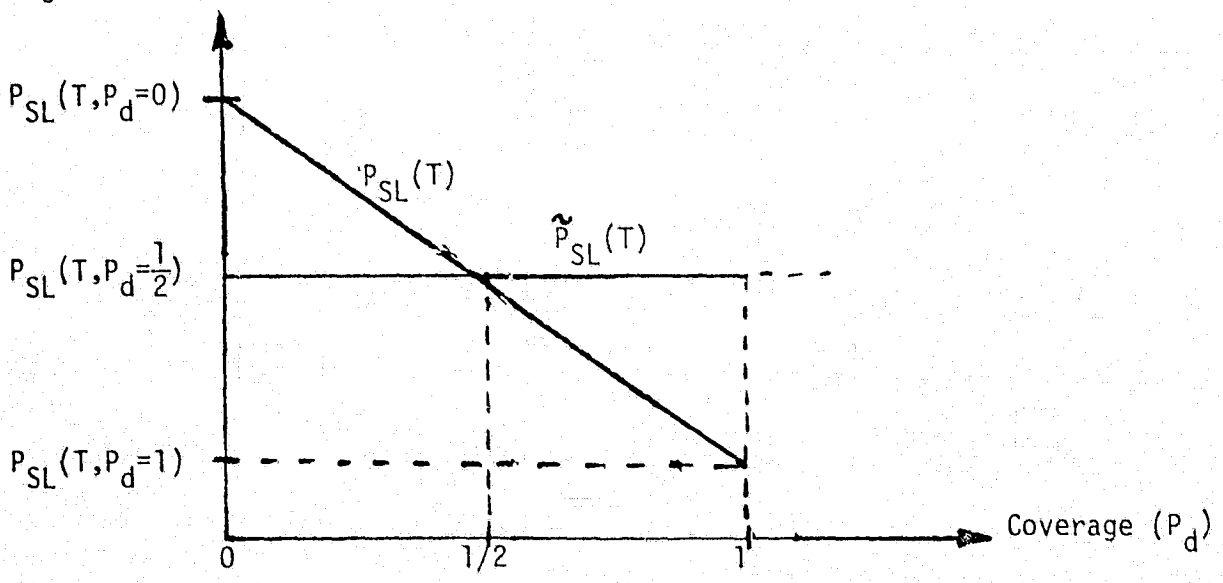

Figure II.4.1

If $P_d = 1$, so that we can detect with probability one a computer failure, when it has occurred, we obtain by (II.4-22) the loss probability to be equal to

$$P_{SL}(T,P_d=1) = (1-e^{-\lambda_c T})^4 \quad . \qquad \text{(II.4-25)}$$

This is the lowest attainable value for the loss probability.

The highest loss probability is observed when $P_d=0$. Then, the self-test techniques are inoperable (or useless), and we have

$$P_{SL}(T,P_d=0) = (1-e^{-\lambda_c T})^3(1+3e^{-\lambda_c T}) \quad . \qquad \text{(II.4-26)}$$

We note that Eq. (II.4-26) incorporates the observation that if $P_d=0$ and two GPCs are left, any GPC failure will result with a system loss condition. Hence, 3 or 4 GPC failures will result with system loss. In turn, $P_d=1$, when two GPCs are left, the system remains operational under a single GPC failure, and is lost only when both GPCs fail. Hence, system loss now occurs only if all 4 GPCs fail yielding expression (II.4-25).

For $P_d = \frac{1}{2}$, we obtain

$$P_{SL}(T,P_d = \tfrac{1}{2}) = (1-e^{-\lambda_c T})^3(1+e^{-\lambda_c T}) \quad . \qquad \text{(II.4-27)}$$

For $P_d = \frac{1}{2}$, we also note that (see (II.4-16))

$$P_{SL}(t,2,P_d=\tfrac{1}{2}) = e^{-\lambda_c t} \qquad \text{(II.4-28)}$$

Consider now the following procedure, to be called the <u>random</u> choice procedure. When two operational GPCs are left, if a failure is observed (through the comparison procedure), one GPC is arbitrarily (at random) shut down. Or, alternatively, when 2 GPCs are left, one GPC is arbitrarily shut down. Under this procedure, the system loss probability, starting with 2 GPCs, $\tilde{P}_{SL}(t,2)$ is obtained to be given by

$$\tilde{P}_{SL}(t,2) = e^{-\lambda_c T} = P_{SL}(t,2,P_d=\tfrac{1}{2}) . \qquad \text{(II.4-29)}$$

The associated system loss probability under a random choice procedure $\tilde{P}_{SL}(T,N)$, is thus equal to that obtained when $P_d = \frac{1}{2}$,

$$\tilde{P}_{SL}(T,N) = P_{SL}(T,N,P_d=\tfrac{1}{2}) . \qquad \text{(II.4-30)}$$

Therefore,

$$\tilde{P}_{SL}(T,N) < P_{SL}(T,N,P_d) \qquad \text{for } P_d < \tfrac{1}{2} , \qquad \text{(II.4-31a)}$$

$$\tilde{P}_{SL}(T,N) > P_{SL}(T,N,P_d) \qquad \text{for } P_d > \tfrac{1}{2} . \qquad \text{(II.4-31b)}$$

Thus, if $P_d < \frac{1}{2}$ the random choice policy is preferrable. Self-test techniques should not then be utilized, since they provide misleading failure information. On the other hand, if $P_d > \frac{1}{2}$, as is the case in the avionic system under consideration, a lower loss probability is attained when self-test techniques are utilized (since they then clearly provide additional helpful failure information).

We now compute the mean duration of the phase-two failure period, denoted by $\bar{T}_F(N,2)$. Phase two starts with two operational GPCs. Let T_1, T_2 denote the lifetimes of these GPCs. These are i.i.d. exponentially distributed random variables with means λ_c^{-1}. The first PGC failure occurs at time $\min(T_1,T_2)$. Then, with probability $1-P_d$ the failure is not detected and the system is lost. With probability P_d, the failure is then detected and the remaining operational GPC continues to operate until it fails. Following these observations, the following result is obtained.

$$\bar{T}_F(N,2) = E\{\min(T_1,T_2)\} + P_d\lambda_c^{-1} ,$$

where

$$E\{\min(T_1,T_2)\} = \frac{1}{2\lambda_c} \quad .$$

Subsequently,

$$\bar{T}_F(N,2) = (2\lambda_c)^{-1} + P_d\lambda_c^{-1}$$

$$= \lambda_c^{-1}(\tfrac{1}{2} + P_d) \qquad \text{(II.4-32)}$$

We again note that under the random choice policy the mean lifetime duration of phase-two, $E(\tilde{T}_F(N,2))$ is given by

$$E[\tilde{T}_F(N,2)] = \lambda_c^{-1} = \bar{T}_F(N,2,P_d=\tfrac{1}{2}) \; . \qquad \text{(II.4-33)}$$

The overall mean lifetime $\bar{T}_F(N)$ is obtained by using Eqs. (II.4-15), (II.4-32), (II.4-33), to be given by

$$\bar{T}_F(N) = \lambda_c^{-1}\left(\sum_{i=3}^{N} i^{-1} + \frac{1}{2} + P_d\right); \qquad \text{(II.4-34)}$$

$$E[T_F(N)] = \lambda_c^{-1}\left(\sum_{i=3}^{N} i^{-1} + 1\right) = \bar{T}_F(N,P_d=\tfrac{1}{2}) \; . \qquad \text{(II.4-35)}$$

In particular, for N=4 the mean lifetimes are obtained by (II.4-34)-(II.4-35) to be equal to

$$\bar{T}_F(4) = \lambda_c^{-1}(P_d + \tfrac{13}{12}) \; ; \qquad \text{(II.4-36)}$$

$$E[T_F(4)] = \frac{19}{12}\lambda_c^{-1} = 1.583\lambda_c^{-1} = \bar{T}_F(4,P_d=\tfrac{1}{2}) \; . \qquad \text{(II.4-37)}$$

For $p_d = 1$, we obtain

$$\bar{T}_F(4,P_d=1) = \frac{25}{12}\lambda_c^{-1} = 2.083\lambda_c^{-1} \quad . \qquad \text{(II.4-38)}$$

The functional dependence of the mean lifetime $\bar{T}_F(4)$ on the coverage probability P_d, indicated by Eq. (II.4-36), is illustrated

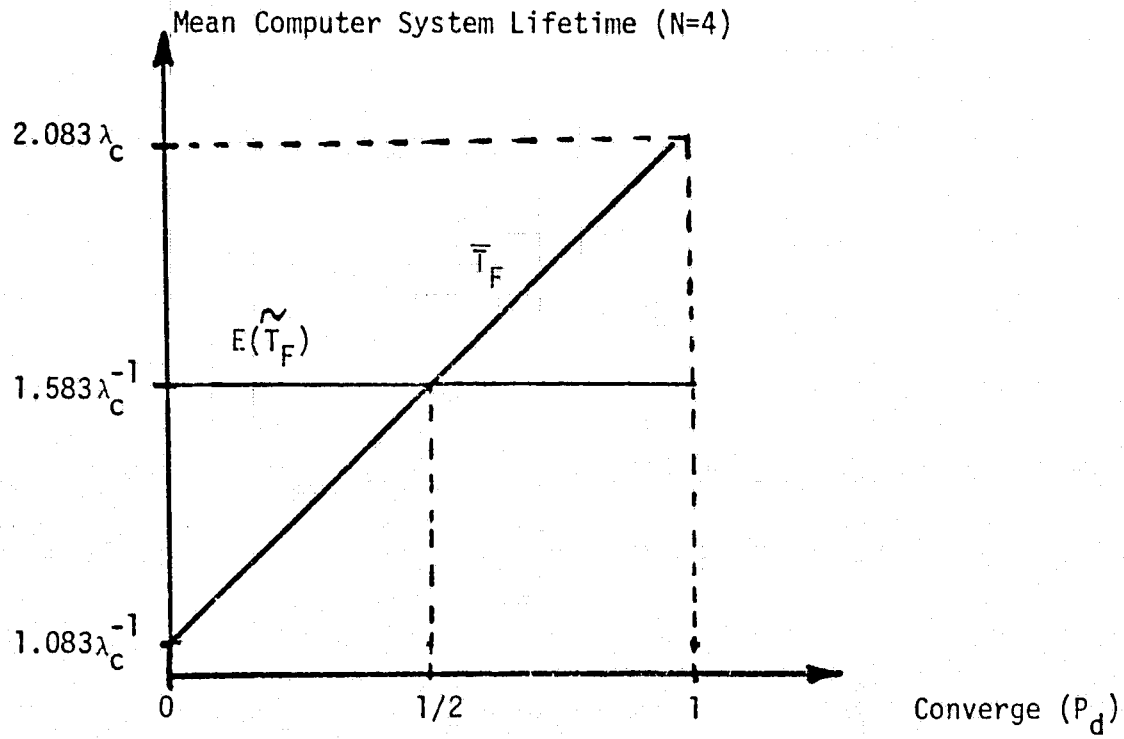

Figure II.4.2

in Fig. II.4.2.

We now examine the dependence of the computer system reliability measures on the number N of parallel computers.

The system loss probability $P_{SL}(T,N)$ is given by formula (II.4-24). If $P_d=1$, we have

$$P_{SL}(T,N,P_d=1) = (1-e^{-\lambda_c T})^N . \qquad \text{(II.4-39)}$$

Therefore, for $P_d=1$,

$$\frac{P_{SL}(T,N+1,P_d=1)}{P_{SL}(T,N,P_d=1)} = (1-e^{-\lambda_c T}) \ ; \qquad \text{(II.4-40)}$$

so that by using an additional parallel PGC we decrease the loss probability by a factor of $(1-e^{-\lambda_c T})^{-1}$.

The mean lifetime $\overline{T}_F(N)$ when N parallel GPCs are used and $P_d=1$, is given by

$$\overline{T}_F(N,P_d=1) = \lambda_c^{-1}\left(\sum_{i=3}^{N} i^{-1} + \frac{3}{2}\right) . \qquad \text{(II.4-41)}$$

Therefore,

$$\frac{\overline{T}_F(N+1,P_d=1)}{\overline{T}_F(N,P_d=1)} = \frac{\frac{3}{2} + \sum_{i=3}^{N+1} i^{-1}}{\frac{3}{2} + \sum_{i=3}^{N} i^{-1}} . \qquad \text{(II.4-42)}$$

Eq. (II.4-42) represents the factor by which the mean lifetime to failure of the redundant computer system is decreased, when the number of parallel GPCs is increased from N to N+1.

For example, if we use only N=3 parallel GPCs, rather than N=4 parallel GPCs, we obtain

$$\frac{\overline{T}_F(3,P_d=1)}{\overline{T}_F(4,P_d=1)} = \frac{\frac{3}{2} + \frac{1}{3}}{\frac{3}{2} + \frac{1}{3} + \frac{1}{4}} = \frac{22}{25} = 0.88 . \qquad \text{(II.4-43)}$$

Thus, using 3 parallel PGCs, rather than 4, reduces the mean lifetime by a factor of 12%.

If we, on the other hand, employ 5 parallel GPCs, rather than 4, we obtain

$$\frac{\overline{T}_F(5,P_d=1)}{\overline{T}_F(4,P_d=1)} = \frac{\frac{3}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5}}{\frac{3}{2} + \frac{1}{3} + \frac{1}{4}} = \frac{137}{125} = 1.095; \qquad \text{(II.4-44)}$$

so that the mean lifetime is increased then by a factor of 9.5%. The system loss probability during an operational period of duration T is then reduced, according to (II.4-40) by a factor of $(1-e^{-\lambda_c T})^{-1}$. The absolute value of the system loss probability is given by (II.4-24).

The equations derived above for the computer system loss probability and lifetime are in terms of the following parameters:

λ_c = the PGC failure rate; T = duration of the redundant phase; N = number of parallel GPCs; P_d = coverage probability. Eq. (II.4-24) yields the loss probability and Eq.(II.4-34) the mean lifetime. The system designer and analyst can use these results to study or adjust the failure and characteristics of the redundant computer system.

II.5 Failure Analysis for an Application Subsystem

We consider an application subsystem of the Space Shuttle avionics data processing network. The failure characteristics of this subsystem are examined in this section.

The subsystem under consideration can be a telemetry subsystem supplying information data to the computer network at certain times; a sensor subsystem; actuator subsystem receiving commands from the computer complex; display subsystem; control subsystem; interface subsystem; GNC subsystem or the mass memory subsystem.

An application subsystem is many times internally redundant. This is teh case for the hand controllers and the keyboard units. Also, all safety-of-flight critical effector subsystems, such as the actuators for the main engine and for the aerosurfaces, the main engine interface units and mission event controllers are internally redundant at different levels. Such subsystems receive redundant commands on separate input channels and using internal algorithms they generate a single output stream. These algorithms also detect incorrect commands and eliminate such commands from consideration in the output.

Subsystems which perform similar functions are assigned to the same data-bus group. Subsystems have different levels of redundancy at the unit level. In accordance with their criticality For example, there are three inertial measurement units, two radar altimeters, and four air data transducer assemblies. To prevent the loss of more than one redundant unit when one data bus fails, no two redundant units interface with the same bus.

To analyze the failure characteristics (invulnerability) of a redundant subsystem, we set the following parameters. The subsystem under consideration is assumed to contain L equivalent redundant units. Each unit is assumed to be connected to a different bus. Thus,

$$L = \text{number of redundant units in the subsystem} = \text{number of data buses connected to the subsystem} \quad \text{(II.5-1)}$$

We characterize the failure properties of each unit by the unit failure rate λ_u,

$$\lambda_u = \text{unit failure rate [failures/sec]} . \quad \text{(II.5-2)}$$

Thus, if T_u is a random variable representing the unit lifetime (i.e., time duration to failure), we have

$$\text{Average time to unit failure} = E(T_u) = \lambda_u^{-1} . \quad \text{(II.5-3)}$$

The unit lifetime distribution is specified as

$$F_u(x) = P(T_u \leq x) , \quad x > 0 \quad \text{(II.5-4)}$$

If the unit lifetime is assumed to be exponentially distributed, we have

$$F_u(x) = 1 - e^{-\lambda_u x} , \quad x > 0 . \quad \text{(II.5-5)}$$

We assume unit lifetimes to be statistically independent and identically distributed. Furthermore, to explicitly illustrate the subsystem failure behavior, we assume now an exponential failure distribution (II.5-5). (The following results, however, are readily extended to include an arbitrary unit lifetime distribution.)

We consider an operational period which lasts for T [sec]. Then we have

$$q_u(T) = \text{probability of a unit failure in T units of time}$$
$$= 1 - e^{-\lambda_u T}. \qquad \text{(II.5-6)}$$

Also,

$$Q_u(T) = \text{probability that all subsystem units fail in T units of time}$$
$$= [q_u(T)]^L = (1 - e^{-\lambda_u T})^L. \qquad \text{(II.5-7)}$$

Each unit is assumed to be connected to a different data bus To evaluate the probability of operational loss (or survival) for the subsystem, we now specify the failure characteristics of the data buses.

Each data bus is associated with a random variable T_ℓ representing its lifetimes (i.e., time to failure). Line failures can be defined to include both physical failures as well as interference (noise) phenomena which cause degradation in data communications across the line. We then set the line failure rate to be

$$\lambda_\ell = \text{data bus (line) failure rate.} \qquad \text{(II.5-8)}$$

The distribution of the line (data bus) lifetime is given by

$$F_\ell(x) = P(T_L \leq x), \quad x > 0. \qquad \text{(II.5-9)}$$

Note that

$$\text{Data bus mean time to failure} = E(T_\ell) = \lambda_\ell^{-1} \qquad \text{(II.5-10)}$$

Assuming the data bus lifetime (time to "failure") to be exponentially distributed, we have

$$F_\ell(x) = 1 - e^{-\lambda_\ell x}, \quad x > 0. \qquad \text{(II.5-11)}$$

The invulnerability of the subsystem is expressed in terms of the following two measures. The subsystem loss probability is defined by

$q_{SL}(T)$ = probability of subsystem loss within T units of time

= probability that within T units of time the subsystem fails or is disconnected from the bus network . (II.5-12)

The subsystem mean lifetime is defined as

$\overline{T}_{SF}$ = the subsystem mean time to failure or disconnection from the bus network (II.5-13)

The subsystem loss probability $q_{SL}(T)$ is computed as follows.

$$q_{SL}(T) = \prod_{i=1}^{L} P\{\text{unit i is lost or disconnected}\}$$

$$= \prod_{i=1}^{L} P \text{ unit fails or its data bus fails}$$

$$= \prod_{i=1}^{L} [1-P(\text{unit i does not fail, its data bus does not fail})]$$

$$= \prod_{i=1}^{L} [1-P(\text{unit i does not fail})P(\text{data bus connected to unit i does not fail})] . \quad \text{(II.5-14)}$$

Therefore, the subsystem loss probability is given by the formula

$$q_{SL}(T) = [1-e^{-(\lambda_u+\lambda_\ell)T}]^L . \quad \text{(II.5-15)}$$

The subsystem mean lifetime (time to failure) $\overline{T}_{SF}$ is similarly derived to be given by

$$\overline{T}_{SF} = (\lambda_\ell+\lambda_u)^{-1} \sum_{i=1}^{L} \frac{1}{i} . \quad \text{(II.5-16)}$$

To derive equation (II.5-16), one notes that if i operating units are left, the time to the next failure (of a unit or its associated

data bus) is exponentially distributed with mean $[i(\lambda_\ell+\lambda_u)]^{-1}$ [sec].

Eqs. (II.5-15)-(II.5-16) provide the desired formula for establishing the failure characteristics of the redundant subsystem. The parameters involved are: the operation period duration (T); the number of redundant units and data buses (L); the failure rate of a unit (λ_u); and the failure rate of the data bus (λ_ℓ). In terms of these parameters, Eq. (I.5-15) yields the probability of subsystem loss (so that no connected operating unit is left), while Eq. (I.5-15) expresses the mean time to system loss.

For given subsystem parameters, these formulas are used to compute the subsystem invulnerability. For a specified subsystem loss probability (or mean lifetime), one uses these results to calculate the desired level of subsystem redundancy and underlying unit and data bus failure rates.

II.6 FAILURE ANALYSIS FOR THE DATA PROCESSING NETWORK

II.6.1 Reliability Performance Measures for the Data Processing Network

The Space Shuttle orbiter avionics data processing network consists of serial data buses which connect the application subsystems to the computer complex. The data buses are divided into groups. Different groups provide communication connections to different subsystems. Certain subsystems contain redundant units, each connected to a different data bus, to increase the subsystem invulnerability to failure.

Reliability measures for the computer system have been presented in Section II.2. The associated failure analysis for the computer system is carried out in sections II.3-II.4. Failure analysis for an application subsystem is presented in Section II.5. In this section we wish to combine these results with the failure characteristics of the data communication network.

The topological structure of the data bus network is specified by the incidence matrix B, where

$$B = [b_{ij}] \qquad \text{(II.6.1-1)}$$

and

$$b_{ij} = \begin{cases} 1, \text{ if data bus } j \text{ connects unit } i \text{ to the computer complex} \\ 0, \text{ otherwise.} \end{cases}$$

Each subsystem contains a number of units. We can thus describe the topological interconnections between the subsystems and the computer complex by a subsystem incidence matrix A, where

$$A = [a_{ij}] \qquad \text{(II.6.1-2)}$$

and

$$a_{ij} = \begin{cases} 1, \text{ if subsystem i is connected to data bus j} \\ 0, \text{ otherwise} \end{cases}$$

The overall network topological structure is specified by the connectivity matrix, also called adjacency matrix, C where

$$C = [c_{ij}] \qquad \text{(II.6.1-3)}$$

and

$$c_{ij} = \begin{cases} 1, \text{ if node i is connected to node j} \\ 0, \text{ otherwise} \end{cases}$$

We regard each network element (GPC, application subsystem or unit) as a node. Nodes are connected by the data bus lines, inducing thus an underlying topological structure modelled as a graph.

Wehn the computer system is in the redundant mode, four GPCs are connected in parallel, having simultaneous access to all application subsystems. We then have

$$a_{ij} = 1,$$

for each unit i and GPC j.

When the computer system is in simplex mode, each subsystem (task) is associated, on a dedicated basis, with a certain computer. Then,

$$a_{ij} = 1$$

whenever subsystem i is associated with GPC j, and $a_{ij} = 0$ otherwise.

We wish to examine the invulnerability of the data processing network to failures of nodes and lines. To assess network reliability, the fo-lowing performance measures are of interest.

We incorporate, as element failures, the failures of computers, data bus lines and subsystem units.

In the redundant mode of operation, we say that a network loss event has occurred whenever a certain set of tasks cannot be processed by the computer complex. This can be due to computer failures, line failures (or noise), or failures of units in certain application subsystems.

The probability of network loss in T units of time is set to be

$P_{NL}(T)$ = probability of network loss is T units of time. (II.6.1-4)

To define and compute $P_{NL}(T)$, we identify a set of critical subsystems (or tasks), the failure of each of which induces a system loss event. We thus set

N_C = set of critical subsystems in the redundant mode. (II.6.1-5)

Subsequently, the network loss probability in the redundant mode is defined as

$P_{NL}(T)$ = probability that, under the redundant mode, a critical subsystem cannot be utilized, or connected to the computer complex, or receive information-processing service from the computer system. (II.6.1-6)

Clearly, in computing $P_{NL}(T)$ we need to consider the availability of computer processing resources to serve the critical subsystem, the reliable transmission of information between the computer complex and the critical subsystems, and the operational integrity of the critical subsystems themselves. We also incorporate the possibility of rerouting upon certain line failures.

In a similar manner, we define the mean time to network loss as

$\overline{T}_{NL}$ = mean time to network loss, under redundant mode
= mean time until the failure of a critical system, or its network disconnection, or the non-availability of computer resources for its associated processing services. (II.6.1-7)

Under a simplex mode of operation, we consider the subnetwork composed of a single GPC and its associated application subsystems. The probability of network loss is then similarly defined as

$q_{NL}(T)$ = probability that a critical subsystem cannot be connected to a GPC in T units of time, under the simplex mode (II.6.7-8)

In computing $q_{NL}(T)$, we consider GPC failures, line failures and unit failures, as before. In addition, we also incorporate the possibilities of rerouting messages (through alternate paths, when their primary paths fail). Also, we consider the utilization of a stand-by GPC to replace a failed computer.

In a similar manner, the mean time to network loss under simple mode is defined by

$\overline{T}_{SNL}$ = mean time to network loss, under simplex mode. (II.6.1-9)

In assessing the interconnecting communication data bus network itself, the following connectivity measures are useful:

K(i) = minimal number of line failures which cause subsystem i to be disconnected. (II.6.1-10)

$P_K(i)$ = probability that subsystem i is disconnected. (II.6.1-11)

For time-critical tasks, it is also of interest to define the delay dependent reliability measure

$P_K(i,D)$ = probability that a task associated with subsystem i cannot be processed by a GPC within D units of time.

(II.6.1-12)

In computing (I.6.1-12), we note that it is possible that the subsystem will remain connected to the computer complex, after certain failures, but due to increased traffic (caused, for example, by rerouting tasks away from failed lines or GPCs), associated tasks cannot receive service (processing) within their required critical time delay constraints.

II.6.2 Failure Analysis for the Data Processing Network: The Redundant Mode

The computer system is assumed to be in the redundant mode. The computer failure rate is λ_c [failures/sec]. The computer coverage probability (i.e., the probability that a GPC will detect its failure, when it has failed, using self-test procedures) is equal to P_d. Then, if N GPCs operate in parallel, the probability of a computer system loss in T sec, $P_{SL}(T,N)$, is given by Eq. (II.4-24) as

$$P_{SL}(T,N) = (1-e^{-\lambda_c T})^{N-1}[1+e^{-\lambda_c T}(N-1-NP_d)] . \qquad (II.6.2-1)$$

The mean time to failure for the computer system is given by (II.4-34) to be equal to

$$\overline{T}_F(N) = \lambda_c^{-1}\left(\sum_{i=3}^{N} i^{-1} + \frac{1}{2} + P_d\right) . \qquad (II.6.2-2)$$

In particular, when N=4, we obtain

$$P_{SL}(T) = P_{SL}(T,4) = (1-e^{-\lambda_c T})^3[1+e^{-\lambda_c T}(3-4P_d)] ; \qquad (II.6.2-3)$$

$$\overline{T}_F = \overline{T}_F(4) = \lambda_c^{-1}\left(P_d + \frac{13}{12}\right) . \qquad (II.6.2-4)$$

Considering now an application subsystem, its failure analysis

has been presented in Section II.5. Assume subsystem i to contain L_i redundant units. Assume each unit to be connected to a single data bus, which is in turn connected to the GPC complex (and thus to all GPCs in the redundant mode). The failure rate of a unit which belongs to subsystem i is set equal to $\lambda_u^{(i)}$ [failures/sec].

The data bus line failure rate is equal to λ_ℓ [failures/sec], for each line. Line failures are assumed to be statistically independent. Time to failure of a data bus line is taken to be governed by an exponential distribution with mean λ_ℓ^{-1}. Then, by Eq. (II.5-15) we find that the probability of subsystem i loss, denoted as $q_{SL}^{(i)}(T)$, indicating the probability that subsystem i will fail or become disconnected within T sec, is given by

$$q_{SL}^{(i)}(T) = [1-e^{-(\lambda_\ell+\lambda_u^{(i)})T}]^{L_i} \quad . \tag{II.6.2-5}$$

The mean time to failure of subsystem i is given, according to Eq. (II.5-16), by

$$\bar{T}_{SF}^{(i)} = (\lambda_\ell+\lambda_u^{(i)})^{-1} \sum_{j=1}^{L_i} \frac{1}{j} \quad . \tag{II.6.2-6}$$

Subsystem i is said to be in a state of network loss if it has failed, is disconnected from the data bus network or if the computer system is lost. We set

$$P_{NL}^{(i)}(T) = \text{probability that subsystem i is in a state of network loss.} \tag{II.6.2-7}$$

Then, combining results (II.6.2-1) and (II.6.2-5), we obtain

$$\begin{aligned} P_{NL}^{(i)}(T) &= 1 - [1-P_{SL}(T,N)][1-q_{SL}^{(i)}(T)] \qquad \text{(II.6.2-8)} \\ &= 1 - \{1 - (1-e^{-\lambda_c T})^{N-1}[1+e^{-\lambda_c T}(N-1-NP_d)]\}\{1-[1-e^{-(\lambda_\ell+\lambda_u^{(i)})T}]^{L_i}\} \end{aligned}$$

If we now let

$$N_c = \{i_1, i_2, \ldots, i_c\} , \qquad \text{(II.6.2-9)}$$

so that subsystems $i_1, i_2, \ldots, i_c$ are regarded as the critical subsystems, then the network loss probability $P_{NL}(T)$ is given by

$$P_{NL}(T) = 1 - \prod_{k=1}^{c} [1-q_{SL}^{(i_k)}(T)]$$

$$\cdot\{1-(1-e^{-\lambda_c T})^{N-1}[1+e^{-\lambda_c T}(N-1-NP_d)]\}. \qquad \text{(II.6.2-10)}$$

Eq. (II.6.2-10) expresses the probability of network survival $1-P_{NL}(T)$, as the product of the survival probabilities of the critical subsystems and the computer system.

The mean time to network loss $\bar{T}_{NL}$ is the time to first failure of the computer system or any one of the critical system, or its disconnection.

Eq. (II.6.2-10) can be used to evaluate the invulnerability of the data processing network to failures of the computer system, data bus lines and application subsystem units.

II.6.3 Network Invulnerability: Alternate Routing and Congestion Effects

The network invulnerability characteristics can be improved by providing alternate routes upon data bus failures. This is demonstrated as follows.

Assume a subsystem with L redundant units. The unit failure rate is λ_u [failures/sec]. The line failure rate is λ_ℓ [failures/sec]. The subsystem is associated with K data buses. A switching capability is provided so that, upon the failure of its line, a unit can be connected to one of the available operational buses associated with

subsystem. Thus, initially each one of the L units is connected to a data bus. When its line fails, a unit can be connected to one of the operational associated lines (including a line that was previously connected to another unit which has failed).

Under such a switching procedure, the probability of subsystem loss, denoted as $\hat{q}_{SL}(T)$ is computed as follows.

$$\begin{aligned}\hat{q}_{SL}(T) &= P\{\text{L units fail or K lines fail, or both}\} \\ &= P\{\text{L units fail}\}+P\{\text{K lines fail}\} \\ &\qquad - P\{\text{L units fail}\}P\{\text{K lines fail}\} \\ &= 1 - [1-(1-e^{-\lambda_u T})^L][1-(1-e^{-\lambda_\ell T})^K] \ . \end{aligned} \tag{II.6.3-1}$$

We note that for $K \geq L$,

$$\hat{q}_{SL}(T) \leq q_{SL}(T) \ . \tag{II.6.3-2}$$

Thus, by providing K alternate data buses, we have decreased the subsystem loss probability.

Such alternate data buses can be provided to the critical subsystems. Providing K_{ij} alternate routes to critical subsystem i_j, we subsequently obtain the network loss probability to be given by (when all routes are assumed to be distinct):

$$\begin{aligned}P_{NL}(T) = 1 - &\prod_{k=1}^{C} [1-q_{SL}^{(i_k)}(T)] \\ &\{1 - (1-e^{-\lambda_c T})^{N-1}[1+e^{-\lambda_c T}(N-1-NP_d)]\} \ , \end{aligned} \tag{II.6.3-3}$$

where

$$1 - q_{SL}^{(i)}(T) = [1-(1-e^{-\lambda_u^{(i)} T})^{L_i}][1-(1-e^{-\lambda_\ell T})^{K_i}] \ . \tag{II.6.3-4}$$

Eq. (II.6.3-3) expresses theprobability $1-P_{NL}(T)$ of network survival as the product of the survival probabilities of the computer complex, critical subsystems and the alternate routes.

In turn, as buses are switched to serve critical tasks, non-critical tasks are delayed. If, however, the number of remaining operational data buses is below a certain critical value m_0, the overall traffic associated with critical tasks is high enough to cause an excessively high message delay value D_0. Under such high message delays, the network cannot provide satisfactory service to the critical tasks, and the network can be said to be lost. This loss probabilty is thus defined as

$\tilde{P}_{NL}(T)$ = probability that the computer system is lost, or a critical subsystem is lost, or that the communication network can provide no more than m_0 interconnecting data buses, causing critical message delay value higher than D_0 . (II.6.3-5)

To compute $\tilde{P}_{NL}(T)$, we model the whole communication network topological structure. We assume that the c critical subsystems can use commonly m data buses, $m \geq c$. Thus, upon the failure of line, an operational line from the pool of these m lines can be rerouted to serve the associated critical subsystem. The subsystems will be disconnected from the computer complex if m-c or more data buses fail. Therefore, we obtain,

Probability of disconnection of critical subsystems from the computer complex in T units of time

$$= \sum_{m-c+1}^{m} \binom{m}{k} (1-e^{-\lambda_\ell T})^k e^{-\lambda_\ell T(m-k)} \quad . \qquad \text{(II.6.3-6)}$$

We need however at least m_0 buses to survive to limit network congestion. Subsequently, the network loss probability $P_{NL}(T)$ is obtained to be given by

$$\tilde{P}_{NL}(T) = 1 - \{1-(1-e^{-\lambda_c T})^{N-1}[1+e^{-\lambda_c T}(N-1-NP_d)]\}$$

$$\left\{\sum_{k=m_0}^{m} \binom{m}{k} e^{-\lambda_\ell Tk}(1-e^{-\lambda_\ell T})^{(m-k)}\right\}\left\{\prod_{k=1}^{c}[1-(1-e^{-\lambda_u^{(i_k)}T})^{L_{i_k}}]\right\} \quad \text{(II.6.3-7)}$$

To explain (II.6.3-7), we note that $1-\tilde{P}_{SL}(T)$ expresses the probability of survival. Then, the first, second and third terms in (II.6.3-7) represent the probabilities of survival for the computer system, communication bus network and the critical subsystems, respectively. The product of the latter terms yields the probability of network survival.

Eq. (II.6.3-7) can now be used to evaluate the data processing invulnerability characteristics, as well as to choose and adjust the underlying failure parameters, topological structure and routing discipline. In particular, we note that the following parameters are involved in computing the network loss probability $P_{NL}(T)$:

- The computer failure rate (λ_c);
- The number of parallel computers (N); (here (N=4);
- The computer coverage probability (P_d); (here P_d=0.96 in redundant mode);
- The duration of operational period under consideration (T);
- The subsystem unit failure rate (λ_u);
- The number of redundant units in a subsystem (L);
- The set of critical subsystems (or tasks, $i_1, i_2, \ldots, i_c$);
- The data-bus line failure rate (λ_ℓ);
- The number of data-bus lines commonly used to interconnect the critical subsystems with the computer complex (m);
- The minimal number of data-bus lines required for a satisfactory interconnection (involving both reliability and congestion performance measures) of the critical subsystem to the computer complex m_0.

Incorporating all these parameters in Eq. (II.6.3-7), we compute the probability $\tilde{P}_{NL}(T)$ of network loss within T units of times. Alternatively, for a prescribed maximal value of $\tilde{P}_{NL}(T)$, we use Eq. (II.6.3-7) to determine the proper computer, subsystem and network (topological) parameters.

We finally note that the network (deterministic) connectivity measures are given as follows.

K = network index of critical connectivity

= minimal number of lines whose failure disconnect the critical subsystems

$$= m-c+1; \qquad \text{(II.6.3-8)}$$

$K(D_0)$ = network index of critical stable connectivity

= minimal number of lines whose failure cause message delay to increase above D_0 sec

$$= m-m_0+1 \qquad \text{(II.6.3-9)}$$

The associated probabilistic connectivity measure is given by $\tilde{P}_{NL}(T)$, and expressed by Eq. (II.6.3-7).

II.6.4 Failure Analysis for the Data Processing Network: The Simplex Mode

Under the simplex mode of operation, tasks and subsystems are divided between two GPCs. The remaining GPCs can then serve as stand-by units.

To characterize system invulnerability to failures of GPCs, data-bus lines and application subsystems, we compute the network loss probability $q_{NL}(T)$, defined by Eq. (II.6.1-8). This function expresses the probability that a critical subsystem under consideration cannot be connected to a GPC, within T units of operational time, under the simplex mode. For that purpose, the

following network structure and parameters are specified.

a) The computer failure rate is equal to λ_c [failures/sec].

b) Two computers need to be in operation. Three computers are initially in a stand-by mode. Upon the failure of a computer, a stand-by GPC is immediately used to replace it, if any operational stand-by computer is available. The computer system is said to be in a state of system loss if there are not two operational GPCs.

c) The computer coverage probability (of failure detection by self-test methods) is equal to P_d.

d) The data-bus line failure rate is equal to λ_ℓ [failures/sec].

e) The subsystem under consideration contains L redundant units. The unit failure rate is equal to λ_u [failures/sec].

f) The subsystem under consideration can use lines taken from a set of m data bus lines. It requires, however, a minimum of m_0 lines, $1 \leq m_0 \leq m$, from this set of m lines, to be able to conduct its information-proce-sing tasks in a satisfactory manner.

g) As an alternative topological model, replacing (f), it can be assumed that the m data-bus lines are shared by m_1 subsystem (or tasks). Each subsystem requires at least a single (distinct) data bus line for its connection to a GPC.

We use the above mentioned system conditions and parameters to evaluate the network loss probability $q_{NL}(T)$. We start by using the study results concerning the failure of the simplex computer system, as presented in Section II.3. By equation (II.3.2-4), the probability

$q_{SL}(T)$ that the simplex computer system will fail in T units of time, when initial 5 GPCs are available, two GPCs are operating simultaneously, and computer system failure is declared when at least four GPCs have failed, is given by

$$q_{SL}(T) = 1 - \sum_{n=0}^{3} e^{-2\lambda_c T} \frac{(2\lambda_c T)^n}{n!}$$

$$= 1-e^{-2\lambda_c T}[1 + 2\lambda_c T + \frac{1}{2}(2\lambda_c T)^2 + \frac{1}{6}(2\lambda_c T)^3] . \qquad \text{(II.6.4-1)}$$

The probability that the application subsystem under consideration will fail, denoted as $q_A(T)$, is obtained by recognizing the latter to fail if and only if all the associated units fail. Therefore, we have

$$q_A(T) = [1-e^{-\lambda_u T}]^L . \qquad \text{(II.6.4-2)}$$

Under assumption (f), the interconnecting data-bus network can serve the underlying subsystem as long as it has m_0, out of m, operating data-bus lines. Therefore, the probability $q_{L,1}(T)$ that the associated interconnecting data-bus network fails, under condition (f), is given by

$$q_{L,1}(T) = \sum_{k=m-m_0+1}^{m} \binom{m}{k}(1-e^{-\lambda_\ell T})^k e^{-\lambda_\ell T(m-k)} . \qquad \text{(II.6.4-3)}$$

Subsequently, the probability $1-q_{L,1}(T)$ that the interconnecting network survives in T units of time (i.e., that it provides a connection between the underlying subsystem and the GPC) is equal to

$$1 - q_{L,1}(T) = \sum_{k=0}^{m-m_0} \binom{m}{k}(1-e^{-\lambda_\ell T})^k e^{-\lambda_\ell T(m-k)}$$

$$= \sum_{k=m_0}^{m} \binom{m}{k} e^{-\lambda_\ell Tk}(1-e^{-\lambda_\ell T})^{m-k} . \qquad \text{(II.6.4-4)}$$

Combining these expressions, we obtain the probability $q_{NL}(T)$ of network loss, under the simplex mode, in T units of time, by writing

$$1-q_{NL}(T) = [1-q_{SL}(T)][1-q_A(T)][1-q_{L,1}(T)] . \qquad \text{(II.6.4-5)}$$

Eq. (II.6.4-5) expresses the probability $1-q_{NL}(T)$ of network survival as the product of the survival probabilities of the simple computer system, the underlying subsystem and the interconnecting data-bus network. Subsequently, substituting (II.6.4-1)-(II.6.4-4) into (II.6.4-5), the network loss probability $q_{NL}(T)$ is obtained to be given by the following formula:

$$q_{NL}(T) = 1 - \{e^{-2\lambda_c T}[1+2\lambda_c T+2(\lambda_c T)^2+\frac{2}{3}(\lambda_c T)^3]\}$$
$$\{1 - [1-e^{-\lambda_u T}]^L\} \left\{\sum_{k=m_0}^{m} \binom{m}{k} e^{-\lambda_\ell T k}(1-e^{-\lambda_\ell T})^{m-k}\right\} \qquad \text{(II.6.4-6)}$$

Using Eq. (II.6.4-5) we can evaluate the network invulnerability to GPC, data-buses and subsystem units, under the simplex mode of operation.

In deriving Eq. (II.6.4-6) we have assumed that the underlying subsystem can employ rerouting procedures in utilizing any one of the operating lines,out of initially available m operating data-bus lines, as long as no less than m_0 data-bus lines are in operation.

Alternatively, to model the sharing of the pool of data bus lines by a number of subsystems, we now assume conditions (g) to hold. Then, m_1 subsystems share the utilization of m data-bus lines. Note, however, that only a single subsystem is allowed to use a certain operational data-bus at one time. (Thus, no time simultaneous use of a data bus by several subsystems is considered.)

Each subsystem requires at least a single (distinct) data bus line for its connection to a GPC. Now, the probability $q_{L,2}(T)$ of failure of the data-bus network, is relative to the subsystem under consideration, is computed as follows.

The data-bus network cannot interconnect the subsystem under consideration if and only if at a certain time, prior to T, the line connected to the subsystem fails, and the number of operational lines then is smaller than m_1 (so that all operational lines are occupied). We set

$$f(u)du = P\{m-m_1\text{-th line failure occurs in } (u,u+du)\}. \qquad \text{(II.6.4-7)}$$

Since, until time u line interfailure times are i.i.d. exponentially distributed with mean $(m_1\lambda_\ell)^{-1}$, we find f(u) to be the Gamma density

$$f(u) = \frac{m_1\lambda_c}{(m-m_1-1)!}\ (m_1\lambda_c u)^{m-m_1-1} e^{-m_1\lambda_c u}, \quad u > 0 . \qquad \text{(II.6.4-8)}$$

The probability $q_{L,2}(T)$ of bus-network loss, relative to the underlying subsystem, is subsequently given by

$$q_{L,2}(T) = \int_0^T f(u)[1-e^{-\lambda_\ell(T-u)}]du . \qquad \text{(II.6.4-9)}$$

Eq. (II.6.4-9) indicates that a bus network loss event will occur if, at some time u, only m_1 lines (out of initial m lines) are left, and in the following T-u units of time the line connecting the subsystem under consideration fails. Substituting (II.6.4-8) in (II.6.4-9) we conclude the result

$$q_{L,2}(T) = \int_0^T \frac{m_1\lambda_\ell}{(n-m_1-1)!}\ (m_1\lambda_\ell u)^{m-m_1-1} e^{-m_1\lambda_\ell u}[1-e^{-\lambda_\ell(T-u)}]du \qquad \text{(II.6.4-10)}$$

As before, the computer system loss probability $q_{SL}(T)$ and the subsystem loss probability are given by Eqs. (II.6.4-1) and (II.6.4-2), respectively. Also the network probability of survival is expressed in accordance with formula (II.6.4-5). We subsequently conclude that the network loss probability udner condition (g), for the simplex mode, denoted as $\tilde{q}_{NL}(T)$, is given by

$$\tilde{q}_{NL}(T) = 1 - \{e^{-2\lambda_c T}[1+2\lambda_c T+2(\lambda_c T)^2 + \frac{2}{3}(\lambda_c T)^3]\}$$
$$\{1 - [1-e^{-\lambda_c T}]^L\}\{1-q_{L,2}(T)\}\,, \qquad \text{(II.6.4-11)}$$

where $q_{L,2}(T)$ is given by Eq. (II.6.4-10).

The mean time to failure of the interconnecting network, relative to the subsystem under considerationis now given by

$$\bar{T}_{NF,2} = \frac{m-m_1}{m_1\lambda_\ell} + \frac{1}{\lambda_\ell} = m\lambda_\ell^{-1}\,. \qquad \text{(II.6.4-12)}$$

In the same manner we derive the formula for the network loss probability when it is assumed that different subsystems (tasks) can share certain data buses on a time division multiplexing (TDM) basis. Then, if we assume that a single data bus can be time-shared among m_T subsystems, (tasks), the following results are obtained.

Under conditions (g), with TDM lines, the data-bus network would not be able to interconnect the subsystem under consideration if and only if at a certain time, prior to T, the line connected to this subsystem fails, and the number of operational lines is smaller than $[m_1/m_T]$; the latter denoting the smallest integer not smaller than m_1/m_T. Therefore, $q_{L,2}(T)$ now is given by Eq. (II.6.4-10) with m_1 there replaced by $[m_1/m_T]$. The network loss probability is subsequently given by Eq. (II.6.4-11) with $q_{L,2}(T)$ expressed as

indicated above.

Finally, we note that incorporating the results of Section II.3.3, one derives in an analogous manner the probability of network loss formulas, under the simplex mode, when restoration procedures are employed to restore failed GPCs.

III. REFERENCES

1. NASA LBJ Space Center, Computer Program Development Specification No. SS-P-0002-110A, Volume 1, Book 1 (revised), Level A Hardware, 26 June 1975.

2. NASA LBJ Space Center, Computer Program Development Specification No. SS-P-0002-120A-1, Volume 1, Book 2 (revised), Level A Software, 20 June 1975 (updated 3 July 1975). Also, SS-P-0002-120C, July 23, 1976.

3. NASA JSC, Computer Program Development Specification No. SS-P-0002-130A, Volume 1, Book 3, Launch Data Bus Software Interface Requirements, 24 June 1975.

4. NASA JSC, Computer Program Development Specification No. SS-P-0002-140, Volume 1, Book 4, Downlist/Uplink Software Requirements, 27 June 1975.

5. NASA JSC, Computer Program Development Specification No. SS-P-0002-410-2, Volume 4, Book 1 (revised), ALT Functional Level Requirements - Guidance, Navigation and Control, 7 July 1975 (updated 25 July 1975 and 8 August 1975).

6. NASA JSC, Computer Program Development Specification No. SS-P-0002-430, Volume 4, Book 3, ALT Functional Level Requirements - System Management Level B, 27 November 1974.

7. Rockwell International Space Division, SD No. 74-SH-0120-06-1-E, Functional Subsystem Software Requirements System Interface, Volume 6, Part 1 of 2, Sections 1 through 11 for Orbiter 101, 4 July 1975.

8. Rockwell International Space Division, SD No. 74-SH-0120-06-1-E, Functional Subsystem Software Requirements System Interface, Volume 6, Part 2 of 2, Appendices A through K for Orbiter 101, 4 July 1975.

9. Rockwell International Space Division, SD No. 74-SH-0230A, Data Processing Subsystem Description and Performance Document, February 1975.

10. IBM Federal Systems Division, 75-A97-001, Space Shuttle Advanced System/4 Pi - Model AP-101, Center Processor Unit, Technical Description, 31 March 1975.

11. IBM Electronic Systems Center, 74-A31-001, Space Shuttle Advanced System/4 Pi - Input/Output Processor (IOP) Principles of Operation: MSC, BCE, MA, and PCI/PCO Functional Description, 6 May 1974.

12. IBM Electronic Systems Center, 74-A31-001, Space Shuttle Advanced System/4 Pi - Input/Output Processor (IOP), Functional Description, 6 May 1974

13. IBM Federal Systems Division, 74-SS-0302A, Space Shuttle Orbiter Avionics Software - Approach and Landing Test (ALT) Functional Design Specification, Volume I, "System Software Overview", 8 November 1974.

14. IBM Federal Systems Division, 75-SS-0714, Space Shuttle Orbiter Avionics Software - ALT Functional Design Specification, Volume II (updated), "System Software", 21 July 1975.

15. IBM Federal Systems Division, 75-SS-0473, Space Shuttle Orbiter Avionics Software - ALT Functional Design Specification, Volume III, "Applications Software", Part 1 - Guidance, Navigation and Control. 17 February 1975.

16. IBM Federal Systems Division, 74-SS-0302A, Space Shuttle Avionics Software - ALT Functional Design Specification, Volume III, "Applications Software", Part 2 - Systems Management, 8 November 1974.

17. IBM Federal Systems Division, 74-SS-0185, Space Shuttle Orbiter Avionics Software - Flight Software Memory Sizing and CPU Loading Estimates, 1 July 1974.

18. IBM Federal Systems Division, 75-SS-0421, IRD No. 1d, Space Shuttle Orbiter Avionics Software - Flight Software Memory Sizing and CPU Loading Estimates, 6 January 1975.

19. Rockwell International, SD74-SH-0120-01-1-NC, Functional Subsystem Requirements Document - System Management, 15 March 1974.

20. Rockwell International, SD74-SH-0120-02-1, Functional Subsystem Software Requirements Document - Flight Computer Operating System, April 5, 1974.

21. Rockwell International, SD74-SH-0120-03-1, Functional Subsystem Software Requirements Document - Guidance and Navigation, 15 March 1974.

22. Rockwell International, SD74-SH-0120-04-1, Functional Subsystem Software Requirements Document - Flight Controls, 15 March 1974.

23. Rockwell International, SD74-SH-0120-05-1, Functional Subsystem Software Requirements Document - Displays and Controls, 15 March 1974.

24. SAMSO, SAMSO-TR74-155, Department of Defense Space Shuttle On-Board Software Requirements, July 1974.

25. System Development Corporation, TN-(L)-5658/000/00, Final Report, Digital Data Processing System, Dynamic Loading Analysis, 30 April 1976.

26. W. T. Chow, "Airborne Computer Technology," Proceedings of the Tenth Space Congress, Cap Canaveral, Florida, April 1973; published by the Canaveral Council of Technical Societies.

27. T. B. Lewis, "Primary Processor and Data Storage Equipment for the Orbiting Astronomical Observatory," IEEE Trans. Electronic Computers, ED-12, 667, 1963.

28. A. E. Cooper and W. T. Chow, "Shuttle Computer Complex," Proceedings of the Sixth Triennial World Congress, IFAC, 1975, Boston/Cambridge, Mass., August 1975.

29. Intermetrics, Requirements and Architecture of the Space Shuttle Flight Computer Operating System FCOS, M3W8XMX-483000, NAS 9-14000, 10 April 1973.

30. J. R. Sklaroff, "Redundancy Management Technique for Space Shuttle Computers," IBM J. Res. Develop. 20, 20, 1976.

31. A. E. Cooper and W. T. Chow, "Development of On-Board Space Computer Systems," IBM J. Res. Develop. 20, 20, Jan. 1976.

32. F. G. Kilmer and J. R. Sklaroff, "Redundant System Design and Flight Test Evaluation for the TAGS Digital Control System," Proceedings of the 29th Annual National Forum of the American Helicopter Society, Washington, D. C., May 1973.

33. H. Hecht, "A Comparison of Fault Tolerant and Externally Redundant Computers," SAMSO TR 74.66, Aerospace Corporation, El Segundo, California, January, 1974 available as document AD777166 from the U. S. National Technical Information Service, Springfield, VA, 22151.

34. E. A. O'Hern, "Space Shuttle Avionics Redundancy Management," presented at the AIAA Digital Avionics Systems Conference, Boston, April 1975.

35. H. A. Padinha, "Divergence in Redundant Guidance, Navigation and Control Systems," Proceedings of the ION National Aerospace Meeting, Washington, D. C., March 1973.

36. D. R. Thomas and F. G. Kilmer, "Redundancy Management Policies for a Dual Redundant Computer Configuration," TR 75-065-0013, IBM Federal Systems Division, Owego, New York, 13827, February 1975.

37. I. Rubin, "Reservation Schemes for Dynamic Packet Access-Control of Multi-Access Communication Channels," Technical Report, UCLA School of Engineering and Applied Science, UCLA-ENG-7712, January 1977.

38. I. Rubin, "Message Delays in FDMA and TDMA Communication Channels," Technical Report, UCLA School of Engineering and Applied Science, in preparation. Also, Proceedings of the Conference on System Sciences, the Johns Hopkins University, Baltimore, MD, March, 1978.

39. I. Rubin, "The Delay-Capacity Product for Store-and Forward Communication Networks: Tree Networks," Appl. Math. & Optim., 2, 197, 1976.

40. I. Rubin, "The Delay-Capacity Product for Message-Switching Communication Networks," J. Combin. Inform. & Syst. Sci. 1, 46, 1976.

41. I. Rubin, "On Reliable Topologies for Computer Networks," Proc. 2nd Intl. Conf. on Software Eng., San Francisco, CA, Oct. 1976.

42. I. Rubin, "On the Design of Reliable Hierarchial Computer Communication Networks," Proc. EUROCON '77, Venice, Italy, May 1977.

43. E. G. Coffman, Jr. and P. J. Denning, Operating Systems Theory, Prentice-Hall, 1973.

44. J. W. Cohen, The Single Server Queue, Wiley, 1969

45. P. Green and R. Lucky, Editors, Computer Communications, IEEE Press.

46. N. Abramson and Kuo, Editors, Computer Communication Networks, IEEE Press.

47. H. M. Goldberg,"Analysis of the Earliest Due Date Scheduling Rule in Queueing Systems," Math. of Op. Res.,Vol. 2, No. 2, May 1977.

48. N. U. Prabhu, Queues and Inventories, Wiley, 1965.

49. T. L. Saaty, Elements of Queueing Theory with Applications, McGraw-Hill, 1961.

50. J. W. Boyse and D. R. Warn, "A Straightforward Model for Computer Performance Prediction," Computing Surveys, Vol. 7, No. 2, June 1975.

51. N. K. Jaiswal, Priority Queues, Academic Press, 1968.

52. H. Goto, "A Digital Phase Lock Loop for Synchronizing Digital Networks," presented at the International Communication Conference, San Francisco, June 1970, pp. 34-21 to 34-25.

53. V. K. Agarwal, "Clock Network Synchronization," TRW IOC 7333.3-55, 1 June 1970.

54. V. K. Agarwal, "Synchronization of Oscillators for Space Shuttle and Other Related Applications," TRW IOC 7352.10-60, Sept. 13,1971.

55. T. J. Stephens, "Shuttle Clock Synchronization," TRW IOC 7333.3-130, 10 March 1971.

56. V. K. Agarwal, "Colocated Transmitters Clock Synchronization," TRW IOC 7352.21-03, March 22, 1972.

57. M. W. Willard and H. R. Dean, "Dynamic Behavior of a System of Mutually Synchronized Oscillators," IEEE Trans. on Comm. Tech., August 1971.

58. M. W. Willard, "Analysis of a System of Mutually Synchronized Oscillators," IEEE Trans. on Comm. Tech., Vol. COM-18, pp. 467-483, October 1970.

59. J. R. Pierce, "Synchronizing Digital Networks," B.S.T.J., 48, No. 3, March 1969, pp. 615-636.

60. I. W. Sandberg, "On Conditions Under Which It is Possible to Synchronize Digital Transmission Systems," B.S.J.T., Vol. 48, pp. 1999-2022, July-August, 1969.

61. H. Inose, et al., "Phase Relation Between Offices in a Mutually Synchronized System," Electronics Letters, 3, No. 6, pp. 243-244, June 1967.

62. H. Mumford, et al., "Synchronization of a PCM Network Using Digital Techniques," Proc. IEE, Vol. 113, No. 9, pp. 1420-1428, September 1966.

63. M. B. Brilliant, "Dynamic Response of Systems of Mutually Synchronized Oscillator," B.S.J.T., pp. 319-356, February 1967.

64. H. R. Krauss, "Clocking and Synchronization within a Fault-Tolerant Multiprocessor," Technical Report T-564, M.I.T., June, 1972.

65. R. W. Larsen, and I. S. Reed, "Redundancy by Coding versus Redundancy by Replication for Failure-Tolerant Sequential Circuits," IEEE Trans. on Computers, Vol. C-21, No. 2, February 1972, pp. 130-137.