RESOURCE ALLOCATION IN COMPUTER NETWORKS

A Thesis
Submitted to the Faculty

of

Purdue University

by

Jie-Yong Juang

In Partial Fulfillment of the
Requirements for the Degree

of

Doctor of Philosophy

August 1985

this is dedicated
to my parents

ACKNOWLEDGMENTS

This thesis could not have been accomplished without the advice and support of Professor Benjamin W. Wah, to whom the author would like to express his most sincere gratitude. The author is also grateful to the other advisory committee members, Professors Edward Coyle, K. S. Fu, Frederic J. Mowle, and Andrew Whinston for their guidance and comments.

This thesis is fortunate to receive consistent support from the National Science Foundation Grants ECS 81-05968 and ECS 80-16580, and CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Milicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW.

TABLE OF CONTENTS

Pag

LIST OF FIGURES

Figure Page

Figure Pag

Appendix
Figure

Page

Page

LIST OF TABLES

ABSTRACT

Juang, Jie-Yong. Ph.D., Purdue University. August 1985. Resource Allocation in Computer Networks. Major Professor: B. W. Wah.

In this thesis, we have studied the resource-allocation problem in resource sharing computer systems. A resource sharing computer system is characterized by a pool of request generators and a pool of resources interconnected by a resource sharing interconnection network. Central issues in resource scheduling include the minimization of resource conflicts, the reduction of the probability of network blockage or congestion, and the balance of workload among resources. Evaluations indicated that distributed state-dependent scheduling schemes are preferable. Moreover, integrating the scheduling schemes into the network protocol significantly reduces the overhead of collecting status information.

A methodology has been proposed to optimize the resource mapping, reduce the scheduling overhead, and facilitate a fast implementation. The methodology has been applied to design resource-allocation schemes for three representative networks of increasing complexities.

For a single contention-bus network, the resource scheduling problem is reduced to the problem of identifying a station with the minimum parameter among a set of physically dispersed random numbers. A distributed minimum-search algorithm that utilizes the collision detection capability of the contention bus has been proposed. The window-search procedure can resolve the global minimum in an average of 2.4 contention steps. No explicit message transfer is required in this process.

For a multiple contention-bus network, the resource-allocation problem i
reduced to the problem of ordered selections. A multi-window search pro
cedure that is an extension of the single-bus search procedure has been pro
posed to select the minimum numbers in parallel. The average time complex
ity of this search procedure is about O($\log_2$t), where t is the number of buses
The mapping between the selected resources and processors has been found t
be the classical stable-marriage problem.

For resource allocation on multistage interconnection networks, the prob
lem is transformed into different network-flow optimization problems, for whic
there exists many efficient algorithms. The network-flow algorithms have bee
integrated into the network protocol with a VLSI systolic-array architectur
that allows resource scheduling to be carried out at signal propagation speed.

CHAPTER I
INTRODUCTION

Due to the rapid advances in microelectronics and Very-Large-Scale-Integrated (VLSI) circuit design technologies, the cost of computer hardware has dropped drastically and the speed of processors has approached some physical limitations. The development of computer networks is also keeping pace with the need of computer communication. The technological advances coupled with the explosion in size and complexity of new applications, have led to the development of resource sharing computer systems. Such systems usually consist of a large number of general and special purpose processors interconnected together by a communication network called *resource sharing interconnection network* [WAH84b].

A resource in a computer network is a processor which performs computation functions or manipulates data objects. It may generate requests to utilize other resources. A system featuring resource sharing capabilities should support *dynamic task migration* which is a scheme that allows tasks to be relocated dynamically. Depending on the system status, a task originally allocated to a processor may be migrated to another processor and executed remotely. Reasons for the migration of a task include (1) the local processor is unable to execute the task because it does not support the designated computing functions; (2) the required data are not available locally and have to be retrieved from a

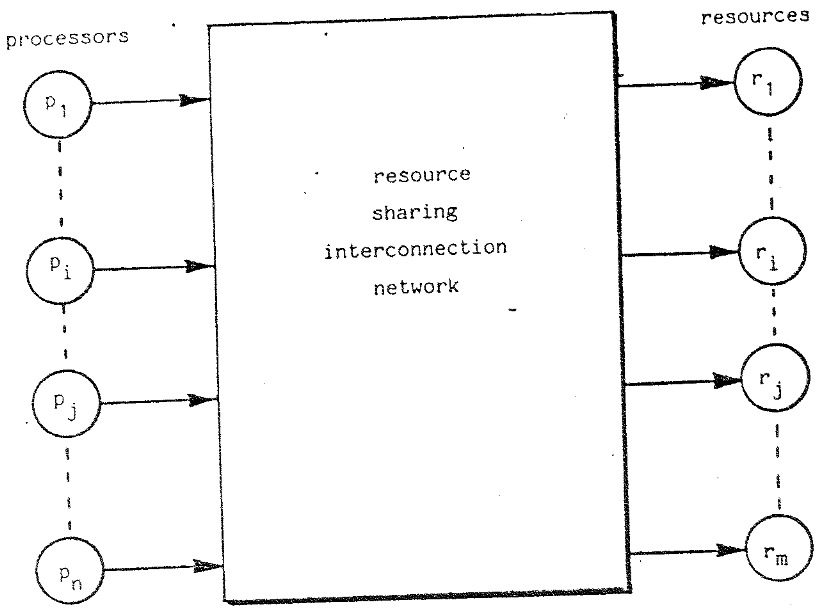
remote site; (3) the workload of the local processor is heavier than that of th
remote processor, i.e., the remote processor has better turnaround time; or (
the local processor is unavailable due to hardware or software failures.

For convenience, the following terms are defined. The processor whic
receives a migrated task is called a ***resource*** to this task, and the processo
from which the task is migrated is called a ***request generator***. The tern
request generating processor and request generator are interchangeable, an
may be abbreviated as request or processor when no ambiguity is aroused.
processor may play both the roles of being a request generator and a resourc
For example, a general purpose processor may request for migrating a Fas
Fourier-Transformation (FFT) task to a special purpose processor. At tl
same time, it may accept a language-translation task from another process
which has much heavier work-load. A special purpose processor can only be
resource if it always receive tasks from the others.

In this chapter, a generic model of resource sharing computer systems
presented. Based on this model, the issues of resource scheduling are explore
and guidelines of designing a good scheduling mechanisms are discussed.

1.1 A Generic Model of Resource Sharing Computer Systems

Resource allocation problem involves the scheduling of resources
resource sharing computer systems. In order to study the resource allocati
problem, a generic model that helps grasping the instantaneous configuration
such systems is devised and shown in Figure 1.1. The model consists of a p



p_i - i'th request generating processor;

r_j - j'th resource for executing a designated task.

Figure 1.1 A generic model of resource sharing computer systems [WAH84b]

of resources and a set of request generating processors. These resources an processors are connected together by an interconnection network. This mode is the logical construction of resource sharing computer systems. A computa tional device that makes requests is represented by a processor in the processo pool, and a computational device which can receive and service tasks i represented by a resource in the resource pool. A device may be either a pro cessor or a resource in this model due to its dual roles of being a resource and request generator. Those devices not involved in resource allocation are no represented in the model.

According to the network selected and the characteristics of processor the proposed model may represent different classes of resource sharing system It includes multiprocessors, local computer networks, or long haul comput networks. The model addresses issues of the global resource allocation ar excludes the details of local scheduling. If a set of tightly coupled processors connected together by a fast network, then the cluster of processors represented by a single request generator or resource in the model. T resource scheduling algorithms developed in this thesis may be applied recu sively to the scheduling within the cluster.

Systems which can be characterized by the generic model have the follo ing properties:

(1) the global status information of the system is not available to the indi dual processors;

(2) the interconnection network is the only inter-communication facil among processors;

(3) a request may be dispatched to any one in the set of available resources which are capable of carrying out the designated task; and

(4) a resource is accessible by any request-generating processors.

A resource sharing system with the above characteristics has the following advantages.

(1) Tasks may be executed in parallel, and workload of processors may be distributed evenly.

(2) Optimized architectures for performing special tasks may be incorporated in the system.

(3) Modifications to include new functions or increased performance can be done easily due to its modularity.

(4) Malfunctional devices may be removed from the system without bringing down the entire system.

Consequently, the performance of such a system may be improved by increasing the number of resources or replacing an existing resource by a more efficient one. The system is also highly reliable and maintainable.

A shared resource may manipulate data objects or provide computational services to a request. Issues on sharing data have been studied intensively in recent years. Many schemes have been proposed to deal with the synchronization of data access and data coherence problems. Examples include monitors and synchronization schemes in operating systems [HOA74, DIJ78], cache coherence schemes in multiprocessors [DUB82], and the methods of maintaining data integrity in distributed database management systems [SEL83]. As we shall see in the next chapter, schemes for sharing computational devices are less

developed. Most existing schemes are based on centralized control or sim
distributed extensions of centralized control. The characteristics of the n
work are usually not incorporated in the design of resource sharing schemes.

The above generic model encompasses many existing or proposed systen
The dynamic task migration is the basic feature of most proposed distribut
programming languages such as the Cooperating Sequential Processes by Ho
[HOA78, SIL83], CLU developed at MIT [LIS81, LIS82], and Distribut
Processes by Brinch Hansen [HAN78]. New operating system designs also p
vides mechanisms to support dynamic task migrations. Examples include pi
in MEDUSA [OUS80] and UNIX [RIT74]. Many architectures also exhibit
characteristics of this generic model. Examples include local computer n
works with load balancing such as the ECN [HWA81] and LOCUS [WAL
VLSI-systolic array multiprocessors [KUN81, KUN82, BRI82], and data-fl
supercomputers [DEN80]. Since resources of these architectures repres
different levels of abstraction, it is instructive to brief these architectures
indicate their mappings to the generic model.

Example 1 : Local Computer Network with Load Balancing

Load balancing is a scheme which engages communication facilities in s
porting remote job execution in a user-transparent manner, so the turnaro
time is reduced through the enhancement of resource sharing. Depending
the workload of processors, the network operating system may distribute j
to a remote processor, or may schedule them for local execution. A local c
puter network with load balancing is illustrated in Figure 1.2 [HWA
WAH83b]. Corresponding to the model, those processors of heavy load

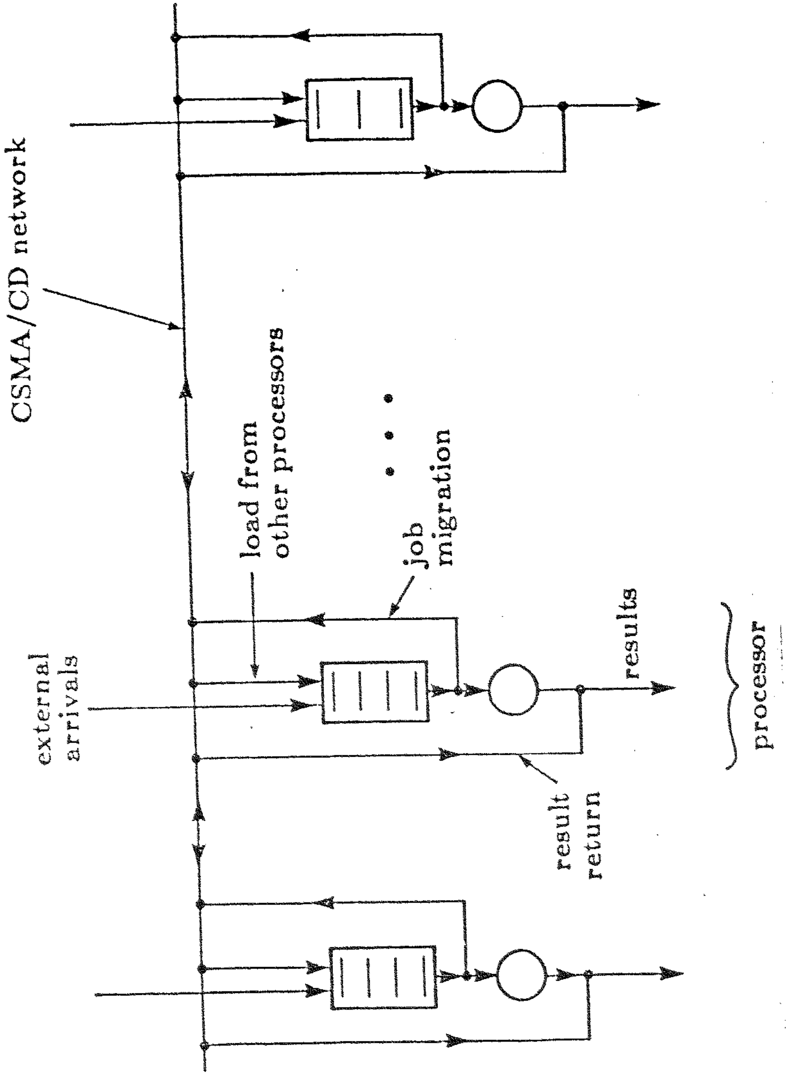


Figure 1.2 A queueing representation of local computer network with load balancing

request-generators, and those processors of light load are resources. Th
resources in this system are job-level processors.

Example 2 : VLSI-Systolic Array Multiprocessors

A VLSI systolic array is a parallel pipeline architecture dedicated to
recursive function, such as the FIR filte:ing, matrix multiplication, or Fas
Fourier Transformation. Such VLSI chips are usually organized as attache
processors to host computers as shown in Figure 1.3 [KUN81]. In this organ
zation, requests are generated from processors and routed to systolic array
through the system bus. The resources in such systems are process-level speci
purpose processors.

Example 3 : Data-Flow Supercomputers

In contrary to the conventional Von Neuman machine, there is r
sequence-control mechanisms in a data-flow machine. The execution of
instruction is driven by the availability of its input data. An instruction
active when all its input arguments are ready. An active instruction is ex
cuted at a processing unit. The outputs of this instruction will activate oth
instructions for the subsequent executions. A data flow program has hi
degree of parallelism and is suitable for running on multiprocessor systems.
typical data flow multiprocessor is shown in Figure 1.4 [DEN80]. In this arc
tecture, instructions are allocated in the activity store and waiting for th
inputs. Once an instruction becomes active, it is routed through an arbitrati
network to a processing element and executed by a processing element. T
output is then routed back to the activity store through a distribution netwo

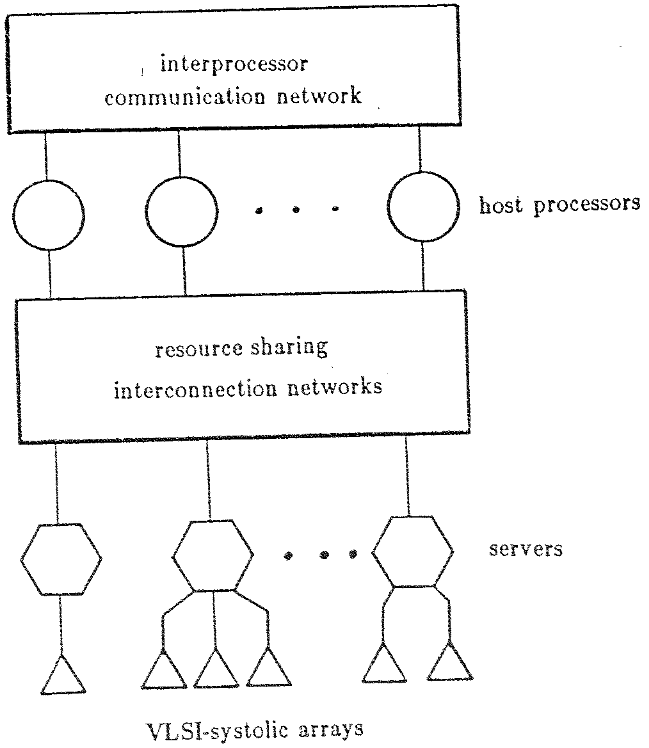


Figure 1.3 An organization of VLSI systolic-array multiprocessors

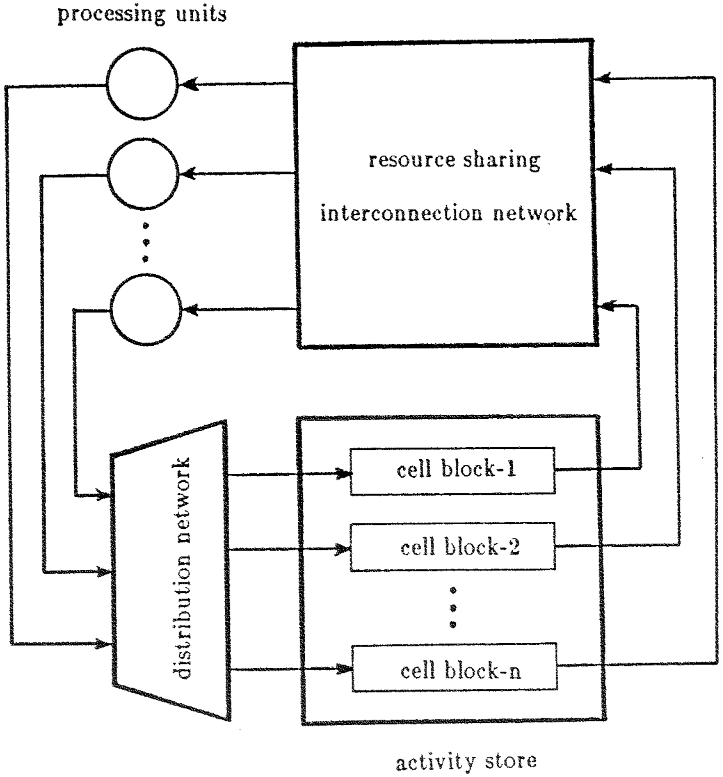


Figure 1.4 A data-flow multiprocessor

The activity store are divided into cell blocks, and active instructions in a cell block are requests. The processing units are arithmetic and logical devices, hence they are instruction-level resources.

1.2 Resource Scheduling

Resource scheduling is to manage the allocation of resources (including communication facilities), so task migrations can be carried out efficiently. In general, the migration of a task in a distributed resource sharing system is divided into three phases: The resource bidding phase, the task migration phase, and the result-return phase. In the first phase, the local processor has to make a request for utilizing a resource. In the second phase, the body of the task, including the task control, program code, and data, are transferred to the resource allocated and executed remotely. In the last phase, the results generated from the execution of the task are routed back to the original processor. Basically, only data transmission is involved during the migration and result-return phases. Resource scheduling is carried out in the resource bidding phase. In this phase, system status information has to be collected, and the decision of resource allocation has to be made.

1.2.1 Issues of Resource Scheduling

The central issue of resource scheduling is to determine a resource mapping that maps requests to resources. Resources will be allocated to associated

requests determined by the mapping. Since a task can be allocated to one
set of free resources, and multiple requests may contend for the same reso
there may be many requests allocated with the same resource while many
ers may be left idle. This problem is called *resource conflicts.* If the resc
has a local memory, the tasks may still be migrated and queued at the resc
regardless of conflicts. This causes no error in operation but may deteric
resource utilization due to the imbalance of workload. If the resource ha
buffering capability, then every request except one has to be rescheduled ag

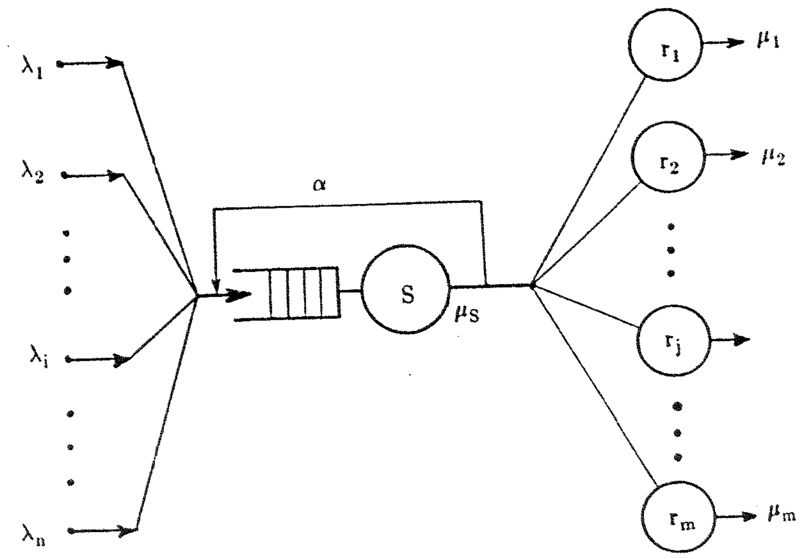
In addition to resource conflict, a bad resource allocation may degrad
performance of the network. Depending on the characteristics of the syste
may have to assign a physical communication link to every processor-resc
allocation and operate in circuit-switching mode, or may share links
operate in packet-switching mode. A resource allocation scheme which mi
izes resource conflicts is not necessarily optimal since there may be many p
being blocked in circuit-switching mode, and packets may be congeste
packet-switching mode.

In summary, there are three major issues that will affect resource ut
tion in resource sharing computer systems. These include network blockin
packet congestions), request conflicts, and imbalanced work-load. Unles
scheduling algorithm is carefully designed and implemented, there ma
many adverse effects on the resource sharing.

1.2.2 Efficiency of Resource Scheduling

To illustrate the effect of the above issues, a resource sharing system may be transformed into a queueing network as shown in Figure 1.5. In this queueing model, a processor is represented by an arrival process with arrival rate λ_i, while a resource is represented by a server r_j with service rate μ_j. An additional server S is introduced to model the resource allocation mechanism and the communication network. A branch from the output of server S feeds back to the input of this server and represents the unsuccessful resource allocations due to network blocking or resource conflicts. Although the scheduling mechanism is represented by a single server, it does not imply a centralized control. Instead, it may be realized in many alternative ways as will be described in Chapter 2. Task transmission delays in the network are considered part of scheduling overhead.

In a word, the service rate of S depends on two factors: the speed of the scheduler and the delay of task transmissions. According to the results of queueing theory [KLE72], the service rate of server S is crucial to the overall system performance. Thus, improving the efficiency of resource allocation is essentially to increase the service rate of the server S and reduce its feedback probability. These may be achieved: (1) by a good design of a high speed resource allocation mechanisms, (2) by utilizing a good scheduling algorithm which generates a good resource mapping, and (3) by using a high-speed communication networks.



S - a server represents the resource allocation mechanism;
α - the probability of network blocking and resource conflicts;
λ_i - request generating rate of processor p_i;
μ_j - service rate of resource r_j;
μ_S - scheduling rate of S.

Figure 1.5 A queueing model for resource sharing computer systems

1.2.3 Scheduling Disciplines

Depending on scheduling disciplines, requests and resources may be characterized by multiple attributes. A request may be represented by the types of task it requests, expected execution time, and priority level. On the other hand, resources may be modeled by its speed, load, and reliability. A scheduling algorithm has to evaluate the allocation costs based on these attributes. .

Consider a scheduling instance that consists two types of requests: matrix computations and scalar computations. Suppose that requests for matrix computations have higher priority than those requests for scalar computations, that all vector processors are busy, and that there are pipelined processors available. Under this condition, a requests for matrix computations would be allocated if the scheduling algorithm determines the priority of requests first, and then assigns resources to the selected requests. It is common that the cost of executing a task that requires many matrix computations on a vector processor would be less than that of running it on a pipelined processor. However, a pipelined processor may be preferable to serve a task with many scalar computations. The cost of allocations would be lower if the scheduling algorithm determines the preference of resource first.

In this thesis, we consider only the class of scheduling disciplines in which multiple attributes are transformed into a single parameter. The parameter that characterizes a request is called the *priority* of the request, and the parameter that characterizes a resource is called the *preference* of the resource.

1.3 The Objective of This Thesis

The objective of this thesis is to investigate the design of efficient resou scheduling mechanisms for resource sharing computer systems, and explore integration of the scheduling algorithms and computer networks. Several go are to be pursued in this research:

(1) A feasible scheduling strategy for improving resource utilization resource sharing computer systems will be studied;

(2) The distribution of scheduling intelligence will be investigated; and

(3) Fast implementation for the scheduling mechanisms will be developed.

A unified design methodology is employed to incorporate the above th design goals. However, we consider only those scheduling schemes that alloc one resource to a request at a time. When multiple resources are requested b single request, they have to be allocated sequentially.

1.4 The Organization of This Thesis

This thesis is presented in the following order. A taxonomy of resource allocation schemes is presented in Chapter 2. This taxonomy class the resource allocation schemes according to the distribution of schedu intelligence and that amount of state information used. Previous work resource sharing systems is also discussed. In Chapters 3 and 4, the opti scheduling algorithms for single contention-bus networks are characterized reduced to the problem of determining the minimum among a set of distrib random numbers. The proposed scheme is integrated into the network by a distributed minimum search algorithm. The extension of the scheduling algorithm for a single contention-bus network to that of multiple contention-bus network is described in Chapter 5. Resource scheduling in multi-stage interconnection networks is discussed in Chapter 6. Lastly, extended applications of the algorithms obtained in this thesis and problems for future studies are identified in Chapter 7.

CHAPTER II
A TAXONOMY OF RESOURCE ALLOCATION SCHEMES

In the design of resource allocation schemes, to achieve high speed schedu ing and to obtain an optimal mapping are usually two mutually conflicti goals. Compromises between the optimality of the scheduling decision and t overhead of the collection of system status information are reached in ma ways. Resource allocation schemes can be characterized by the trade between these two goals. In this chapter, a taxonomy of these resource alloc tion schemes is presented. The advantages and disadvantages of each class resource allocation schemes in the taxonomy are explored. This leads to t conclusion that distributed state-dependent-allocation scheme is a prefera resource allocation mechanism. To tackle the complex design problems of d tributed state-dependent resource allocation schemes, a systematic desi methodology is proposed in this chapter.

2.1 A Taxonomy of Resource-Allocation Schemes

A taxonomy that categorizes most resource allocation schemes is given Figure 2.1. Resource allocation schemes may be classified into two clas depending on whether global status information is used or not, and whet

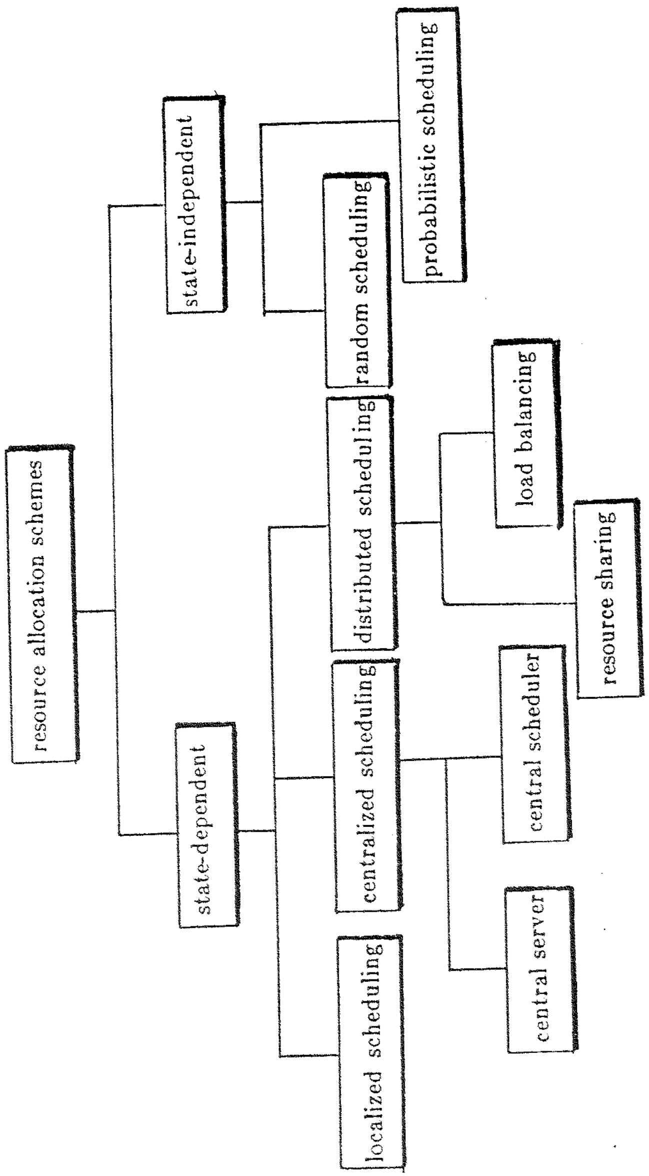


Figure 2.1 A taxonomy of resource-allocation schemes

they are *state-dependent* or *state-independent*.

In the class of state-independent scheduling schemes, resource allocation i carried out by the individual request-generator. The processor determine which resource to bid for based on the local information available. The infor mation available may include the statistics of the system's operating histor which is usually characterized by a stochastic process, piggy-backed informa tion carried by the return message of previous requests, and the specificatior of individual resources. If the processor chooses a resource randomly, then tl scheduling scheme is a *random scheduling* scheme. On the other hand, if tl statistics on previous requests and resource specifications are inputs to tl scheduling decision then the scheduling scheme is a *probabilistic schedulii* scheme. Since requesting processors do not communicate, resource conflic are unavoidable. Conflicting requests have to contend for the resource, th bid for. Queueing-network analysis has been applied intensively to analyze t efficiency of this class of scheduling schemes [TOW80, NI81, WAN85].

On the other hand, in the class of state-dependent scheduling schemes, t global status information is crucial. Among them, a *localized state-depende scheduling* scheme requires every processors to maintain a copy of the glol state information and to determine the resource allocation independent While in *centralized state-dependent scheduling* schemes, the status informati is collected by a central control node which determines the resource mapping be distributed to all requesting processors. A scheduling scheme with this ki of organization is called a *central scheduler*. Essentially, only the tasks resource bidding are carried out in the central control node. Task migratic and result-returns are carried out independently. However, the central cont node may be also responsible for buffering tasks and dispatching them to resources. In this sense, the scheduler becomes a *central server*. A central server is usually found in systems in which resources do not have local memories. Some master-slave multiprocessors belong to this class [ENS77].

A *distributed scheduling* scheme differs from the localized scheduling and centralized scheduling schemes in the way that global status information are collected and utilized. In a distributed scheduling scheme, only partial status information is maintained by each processor, and the scheduling decision is made cooperatively through exchanging information. The amount of information flow is usually lower than that of the previous two approaches.

Most existing resource scheduling schemes belong to the class of state-independent schemes [DAY79, JAY82, LEI81, LEI84]. The resource sharing protocol of ARPANET is a typical example in which task migrations are determined by end users[THO73]. This class of resource scheduling schemes is simple and incurs relatively little overhead. Nevertheless, the problem of low resource utilization remains unsolved. Centralized state-dependent scheduling schemes can be found in many multiprocessor systems with master-slave structure [ENS77, HAY78, BAE80, HWA84]. However, adopting this approach to distributed systems tends to eliminate their advantages. Localized state-dependent scheduling schemes are the direct distributed extensions of centralized control. The load balancing schemes of ECN and LOCUS belong to this class. Although this approach can be implemented in an existing network, it incurs a large amount of redundant information flow and is hard to maintain a consistent state information due to the network delay. Consequently, a resource mapping generated by an optimal scheduling algorithm is not

necessarily the optimal one since inaccurate information may be used.

Distributed state-dependent scheduling schemes are generally preferab for the following reasons: (1) The information flow in maintaining the glob information is reduced because status information is utilized efficiently; (They can achieve optimal resource allocation; and (3) Their speed may l increased due to the concurrent execution of scheduling tasks. Only a few sin ple distributed state-dependent schemes have been proposed [MAN84, RAT8 WAH82, WAH83b, WAH84b, JUA84b-c]. It is still an open area to study. W focus on the design of distributed state-dependent resource scheduling schem in this thesis.

2.2 Implementation Considerations and A Design Methodology

In general, a resource scheduling algorithm generates a resource mappi according to the system status information. A good resource mapping is o that minimizes a cost function under the network constraints. The cost fur tion is usually determined by the scheduling disciplines. It is usually easier optimize the cost function regardless the constraints imposed by the netwo As a result, many processors may not be allocated a scheduled resource due conflicts in the network. To reduce this probability, A high-bandwidth n work is usually used [FEN81]. A crossbar network has been used in syste such as the C.mmp[ENS77, FUL78]. It does not have network blocking pr lem. However, the cost of a crossbar network is O(n^2) , where n is the numl of devices connected, and is not practical when the system is large.

multistage interconnection network is a cost-effective choice [WAH84b], but blocking probability may be as high as 60% [PAT81, FRA82] if resources are not allocated properly. These observations indicate that a good resource scheduling algorithm should incorporate network constraints in the optimization of a given scheduling discipline. The following design methodology is proposed.

(1) Formulate into a constrainted optimization problem.

(2) Design a distributed algorithm to solve the problem.

(3) Identify primitive operations of each process in the distributed algorithm.

(4) Integrate the primitive operations into the network.

The cost of collecting status information may also be included in the objective function, so tradeoffs can be made between the amount of status information used and the efficiency of the scheduling algorithm. A well-designed distributed algorithm should reduce unnecessary message passing. The crucial speedup of the scheduling scheme lies in implementing the primitive operation into the network. This approach essentially shifts the responsibility of scheduling requests by the request generators to the network

We have applied the methodology to the design of resource scheduling schemes on three types of networks: single contention-bus network, multiple contention-bus networks, and multistage interconnection networks. They represent classes of networks of increasing complexities. In each case, we have obtained optimal scheduling algorithms with efficient decision-support mechanisms. The resource scheduling algorithms obtained is fast and incurs small network blockages.

CHAPTER III
RESOURCE ALLOCATION IN SINGLE CONTENTION BUS

In this chapter, resource allocation schemes on a single-bus system ar developed. A special class of resource allocation schemes that encompasses mos of the conventional scheduling disciplines are characterized. An optima scheduling algorithm for this class of allocation schemes on a bit-serial contention bus is also developed. The scheduling algorithm is obtained by applyin our proposed design methodology in Chapter 2. The architecture and the basi operations of a contention bus are reviewed first, followed by the characteristic of the optimal algorithm. A distributed scheduling algorithm is then proposed and the integration of the scheduling algorithms into the network is described.

3.1 Single Contention Bus Networks

There is only one communication channel in a single bus system, an only one processor is allowed to transmit at a time. Processors in the networ communicate with each other by passing messages through the bus. Message transmitted over the bus are broadcast to every processor in the network. diagram of such a network is depicted in Figure 3.1a.

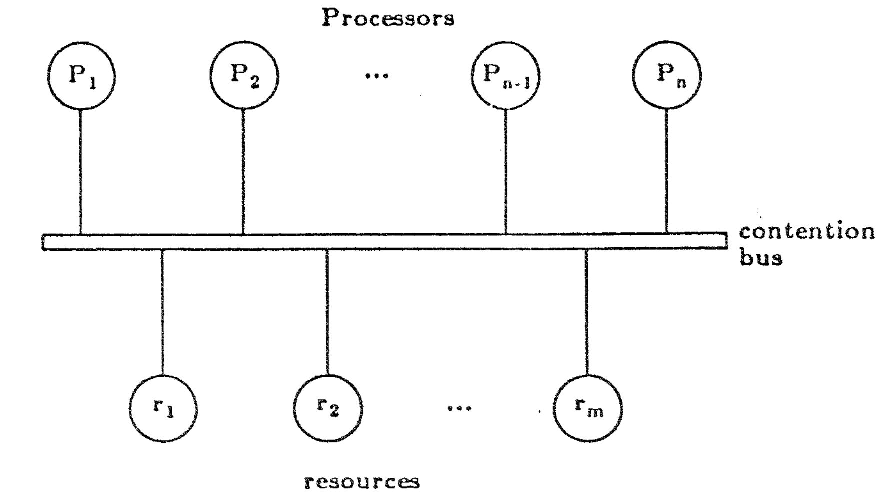


Figure 3.1a A resource sharing system connected by a single contention bus.

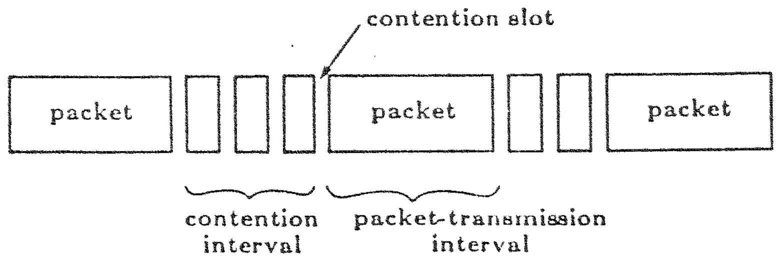


Figure 3.1b The operations of a contention bus with alternating phases

A contention bus is a bus in which a distributed random-access bı arbitration scheme is used. The operation of the bus is divided into two alt nating phases, contention-resolution phase and data-transmission phase, whi are illustrated in Figure 3.1b.

The contention-resolution protocol is shown in Figure 3.2. Whenever processor has data ready to transmit, it checks the bus status to see if the b is free. If the bus is in use, the processor has to wait until it becomes fr Once the bus is sensed free, a processor is allowed to transmit its data. T message transmitted will propagate along the bus to its destination. Befc this message arrives, other processors also detect the same status and transn their data. Consequently, data transmitted from different sources will interfe with each other, and would not be recognizable. Should this happens, a *co sion* is said to occurs. In a contention network, every processor is able detect the occurrence of collisions. This capability is called *collision detecti* In the worst case, it takes an end-to-end propagation time for every process to detect the occurrence of a collision. This period is called the *collisio detection time* or a *contention slot*. During the collision-detection time, transmission is vulnerable. A processor cannot be sure that its curre transmission is successful until it has transmitted for a time longer than collision-detection time without any collision. To avoid repeated collisions, contention-resolution protocol is used to control transmissions.

Many contention-resolution protocols have been proposed and imp mented during the last decade [KLE75, MET76, CAP79a, CAP79b, HLU8 Basically, they follow the structure shown in Figure 3.2 and are distinguish by the different transmission controls.

TYPICAL CONTENTION RESOLUTION PROTOCOL

• The Flow Chart :

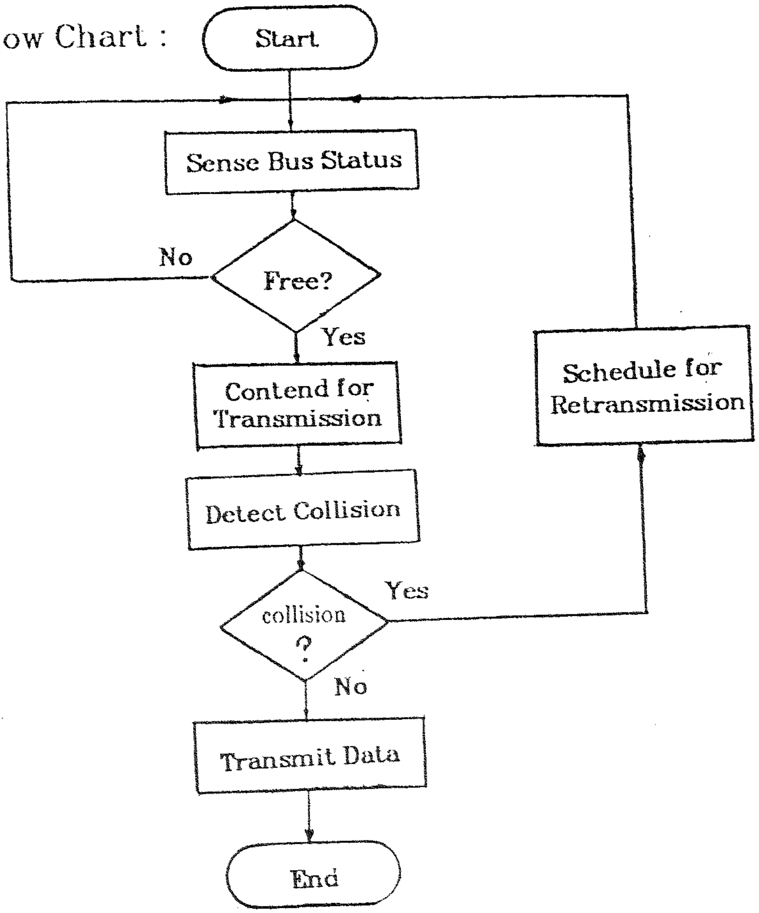


Figure 3.2 A typical contention resolution protocol showing the transmission of one message

In summary, a single-contention bus network is characterized by the fol
lowing properties: (1) there is a single communication channel; (2) every proces
sors on the bus can receive data simultaneously; (3) random access is allowed
(4) the processors are capable of detecting collision; and (5) a distribute
transmission control is used.

3.2 The Optimal Resource Allocation Algorithms

The objective of a resource scheduling algorithm is to generate a
appropriate resource mapping. As discussed in Chapter 1, an optimal resourc
mapping for a circuit-switched network may maximize the number of resourc
allocated and minimize the allocation costs. If the system operates in a packe
switched mode an optimal mapping is one that balances the workload
resources and minimizes the transmission delays.

In general, the optimal resource allocation problem is a constrained optim
zation problem that optimizes the system performance subject to the co
straints of the network. Since there is only one communication channel in
single-bus system, only one resource can be allocated at a time. Therefore, t
scheduling problem is reduced to finding a pair of processor and resource th
optimize the system performance. In a scheduling algorithm for load balancin
the central problem is the determination of a request-generator from which
task is to be migrated and a resource to service the request, so the avera
response time of servicing jobs is minimized.

Let Q be the set of request generators and R be the set resources. The priority level of a request p ∈ Q is x_p, and the preference level of a resource s ∈ R is y_s. Due to the network constraint that only one job can be transmitted at any time, the optimization of resource scheduling on a single bus may be represented by the following non-constrainted optimization problem:

$$\min_{(p,s)\in Q\times R} H(x_p,y_s) \tag{3.1}$$

The cost function H is defined with respect to different scheduling disciplines. It may be very complex and difficult to optimize in general. We will only study a special class of the cost functions that satisfy the following conditions.

$$\frac{\partial}{\partial x_p} H(x_p,y_s) \leq 0$$

$$\frac{\partial}{\partial y_s} H(x_p,y_s) \leq 0 \tag{3.2}$$

Eq. 3.2 implies that the cost of utilizing a resource is independent of the characteristics of the task, and the cost of dispatching a task is independent of the resource allocated. Most existing scheduling problems, such as load balancing and FCFS, can be solved by independently selecting the task to be serviced and the resource to service the task. Hence, the independence relations hold for these problems.

It follows directly from Eq's 3.1 and 3.2 that

$$\min_{(p,s)\in Q\times R} H(x_p,y_s) = H(\ \max_{p\in Q}(x_p),\ \max_{s\in R}(y_s)\ ) \tag{3.3}$$

According to Eq. 3.3, resource scheduling may be split into two steps. A request

p which maximizes x_p and a resource s which maximizes y_s can be determine independently in each step. This property allows the problem of resourc scheduling in a single bus system to be reduced to the problem of determinin the maximum among a set of random numbers called ***contention parameters***.

The generation of contention parameters may be dependent on each othe and may be site-dependent. For tractability reason, the parameters a assumed to be independently generated and possibly site-dependent in th thesis.

3.3 A Distributed Minimum-Search Algorithm

Conventionally, the implementation of an extremum-search algorithm in distributed system relies on the message-passing mechanism to collect all tl random numbers to a central site. It needs O(n) messages, where n is tl number of processors. In this section, an efficient distributed algorithm f identifying the minimum is presented. The algorithm for searching the ma imum is similar. In this chapter, we will describe an efficient protocol for bit-serial contention bus to determine the extremum of a set of rando numbers. A corresponding scheme for a bit-parallel bus will be discussed Chapter 4. It is assumed that each processor is able to perform the followi two operations: (1) maintain a global reference interval or *window*, and count whether there is none, one, or more than one random number falling the reference interval.

Based on these operations, the minimum of a set of distributed random numbers can be searched in the following way. Suppose the set of contention parameters are $\{x_1, ..., x_n\}$ in the interval between L and U, and y_i is the i-th smallest of the x_j's. To search for the minimum, an initial interval or *window* is chosen with the lower bound at L and the upper bound between L and U. There can be zero, one, or more than one number in this window. If there is exactly one number falling within the window, this number can be verified as the minimum, y_1. Otherwise, the window has to be updated: it is moved if it is empty, or is shrunk to a smaller size if it contains more than one number. The process is repeated until the minimum is uniquely isolated in the window. This algorithm is called the *window-search algorithm*. An example of the steps involved are shown in Figure 3.3.

3.4 Integration on A Bit-Serial Single Contention Bus

The distributed minimum search scheme described in the last section incurs no explicit message transfer because every processor can maintain the global window and are able to count the number of random numbers in it. In a conventional distributed system, the algorithm can be implemented on top of the message-transfer mechanisms. The global reference interval can be maintained with the aid of synchronization schemes that are realizable by message passing. Counting numbers can also be done by message passing . The message communication overhead can be reduced if a collision detection mechanism is available.

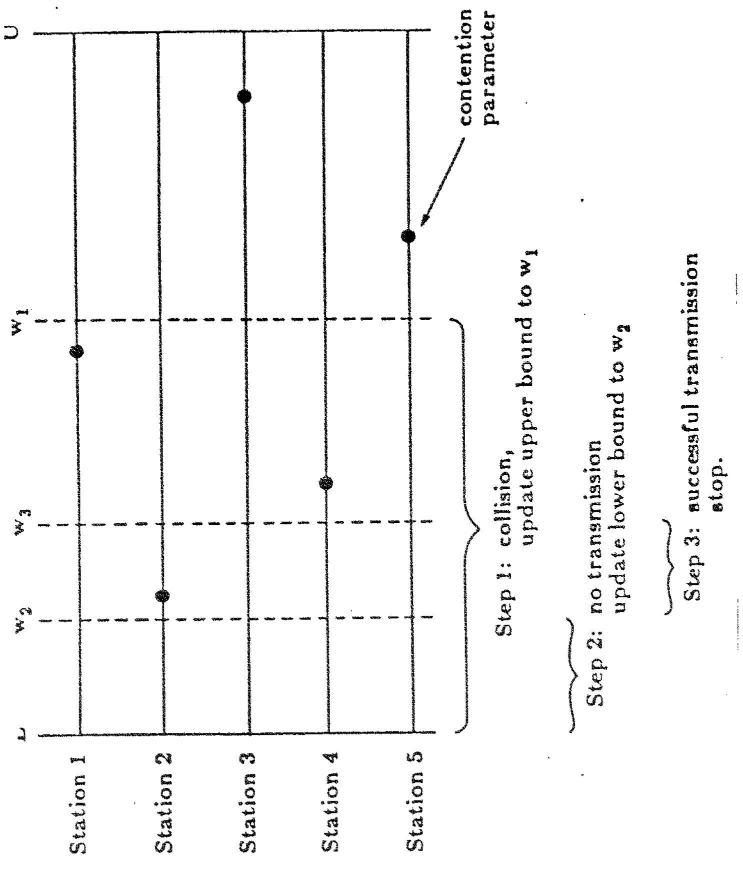

L
w_2
w_3
w_1
U
Station 1
Station 2
Station 3
Station 4
Station 5
contention parameter
Step 1: collision, update upper bound to w_1
Step 2: no transmission update lower bound to w_2
Step 3: successful transmission stop.

Figure 3.3 An example illustrating the updates of the global window to isolate the station with minimum contention parameter (Braces indicate windows used in different steps)

(a) Counting random numbers:

This operation can be implemented with the collision detection capability, which is a primitive operation in contention-bus networks. A no-transmission in a contention slot implies that none of the numbers is in the window; a successful transmission indicates that there is exactly one number in the window; and a collision shows that there are more than one number in the window.

(b) Synchronizing the global window:

Instead of synchronizing the global window by explicit message passing, this operation may be done by running a common algorithm with identical inputs. Since identical information is received by all stations in each contention slot, the global window can be synchronized by the common window-update rule as described in Section 3.5.

An implementation of the distributed window-search algorithm at Station i, $1 \leq i \leq n$, on a multiaccess bus with a three-state collision-detection mechanism is shown in Figure. 3.4.

3.5 Window Control Algorithms

In general, the update of the global window has two degrees of freedom: position and size. The determination of its position has to be constrainted, so the processor that wins the contention is the one that owns the minimum. This can be done by the window-update rule as described in the window-search algorithm. The determination of size will greatly affect the performance of the algorithm. Four window-control algorithms are described below. As discussed

```
procedure window_protocol_station_i;
/* procedure to find window boundaries for isolating one of the contending stations */
[ /*   window - function to calculate window size w,
       random - function to generate local contention parameter,
       estimate - function to estimate channel load,
       transmit_signal - function to send signal to bus with
               other stations synchronously,
       detect - function to detect whether there is collision on the bus (three-state),
       r_i - local contention parameter,
       n̂ - estimated channel load,
       lb_minimum - lower bound of interval containing minimum (minimum is L),
       ub_minimum - upper bound of interval containing minimum (maximum is U),
       contending - boolean to continue the contention process,
       state - state of collision detect, can be collision, idle or success
               (for three-state collision detection). */

       lb_minimum := L;
       ub_minimum := U;
       r_i := random (L,U);
       n̂ := estimate ();
       w := window (lb_minimum, ub_minimum, n̂);
       contending := true;

       while (contending) do [
               if (r_i ≥ lb_minimum and r_i ≤ w) then [
                       transmit_signal ();
                       /* test for unique station in the window */
                       state := detect ();
                       if state = idle then
                               /* update lower bound of interval containing minimum */
                               lb_minimum := w;
                       else if state = collision then
                               /* update upper bound of interval containing minimum */
                               ub_minimum := w;
                       else  /* successful isolation of minimum */
                               return (lb_minimum, ub_minimum);
                       w := window (lb_minimum, ub_minimum, n̂) ]
               else
                       contending := false   /* stop contenting */
       ]
       return (failure)
]
```

Figure 3.4 A window protocol for determining the minimum of distribut random numbers

before, the inputs to these algorithms can be collected from information broadcast on the bus.

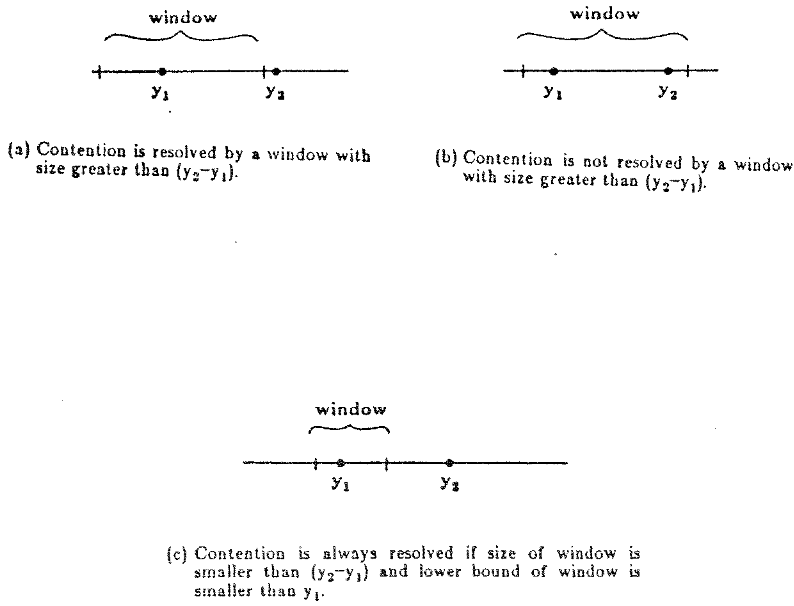
3.5.1 Binary-Divide Window Control

A straightforward way to determine the window size is to choose the upper bound of the window at the middle of previous window. It is interesting to investigate the performance of this simple method because it provides a lower bound on the performance.

The overhead is analyzed in terms of the number of contentions the protocol to determine the minimum. In any given step, if the window size is greater than the distance between the two smallest random numbers, y_1 and y_2, then the minimum may be isolated depending on the relative positions of y_1, y_2, and the window (Figures 3.5a and 3.5b). On the other hand, if the window is reduced to a size smaller than the distance between y_1 and y_2, and the bounds of the window are updated according to the procedures in Figure 3.4, then the minimum will always be isolated in such a window. This is illustrated in Figure 3.5c. Hence the maximum number of contentions required to resolve the minimum never exceeds the number of steps required to reduce the window to a size smaller than the distance between y_1 and y_2. Assuming that k steps are required, the following condition holds :

$$2^{-k} < y_2-y_1 < 2^{-(k-1)} \tag{3.4}$$

Taking the logarithm of the inequality in Eq. 3.4 and rearranging it,



(a) Contention is resolved by a window with size greater than (y_2-y_1).

(b) Contention is not resolved by a window with size greater than (y_2-y_1).

(c) Contention is always resolved if size of window is smaller than (y_2-y_1) and lower bound of window is smaller than y_1.

Figure 3.5 Possible sizes and positions of a window during a contention

$$-\frac{\log_e(y_2-y_1)}{\log_e 2} < k < 1-\frac{\log_e(y_2-y_1)}{\log_e 2} \qquad (3.5)$$

This inequality gives the upper bound of the binary-divide window control for a given instance of y_1 and y_2. From the theory of ordered statistics [DAV70, FEL71], if the y_i's are uniformly distributed over the interval (0,1), then the joint probability density function of y_1 and y_2 is

$$f_{y_1y_2}(y_1,y_2) = \begin{cases} \frac{n!}{(n-2)!}(1-y_2)^{n-2} & \text{for } 1 \geq y_2 > y_1 \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad (3.6)$$

From Eq.'s 3.5 and 3.6, E(k), the average number of contention slots to resolve contentions in the binary-divide window control, can be obtained by integrating the weighted upper bound over the domains of y_1 and y_2.

$$E(K) < \int_0^1\int_0^{y_2}\left[1-\frac{\log_e(y_2-y_1)}{\log_e 2}\right]\frac{n!}{(n-2)!}(1-y_2)^{n-2}dy_1dy_2$$

$$< 1-\frac{n!}{(n-2)!\log_e 2}\int_0^1\left[\int_0^{y_2}\log_e(y_2-y_1)dy_1\right](1-y_2)^{n-2}dy_2 \qquad (3.7)$$

Since $\int_0^{y_2}\log_e(y_2-y_1)dy_1 = y_2\log_e y_2 - y_2$. Eq. 3.7 can be simplified as:

$$E(K) < 1-\frac{n!}{(n-2)!\log_e 2}\int_0^1(1-y_2)^{n-2}(y_2\log_e y_2 - y_2)dy_2 \qquad (3.8)$$

The integration in the RHS of Eq. 3.8 can be evaluated to become:

$$\int_0^1(1-y_2)^{n-2}y_2\log_e y_2dy_2 = H_n - H_{n-1} \qquad (3.$$

where H_n is the harmonic mean of the series {1,2, ... , n}, i.e.

$$H_n = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{i} \qquad (3.1$$

The harmonic mean is approximately equal to $\frac{\log_e n + \gamma + \frac{O(1)}{n}}{n}$ [REI7

where γ is a constant. Hence, from Eq.'s 3.7 thru 3.10, we obtain

$$E(k) < 1-\frac{n(n-1)}{\log_e 2}\left[\left(\frac{\log_e n}{n} - \frac{\log_e(n-1)}{n-1}\right) - \left(\frac{1}{n-1} - \frac{1}{n}\right)\right] \qquad (3.1$$

Since $\log_e n \simeq \log_e(n-1)$ for large n, Eq. 3.11 may be reduced to:

$$E(k) < 1 + \frac{n(n-1)}{\log_e 2}\frac{(1+\log_e n)}{n(n-1)} < 3 + \log_2 n \qquad (3.1$$

Hence,

$$E(K) = O(\log_2 n). \qquad (3.$$

In addition to the above analysis, a simulation has also been conducted evaluate the performance of the binary-divide window control. The results plotted in Figure 3.6. Note that the average number of contention slots smaller than $O(\log_2 n)$, which confirms that $O(\log_2 n)$ is an upper bound on average performance.

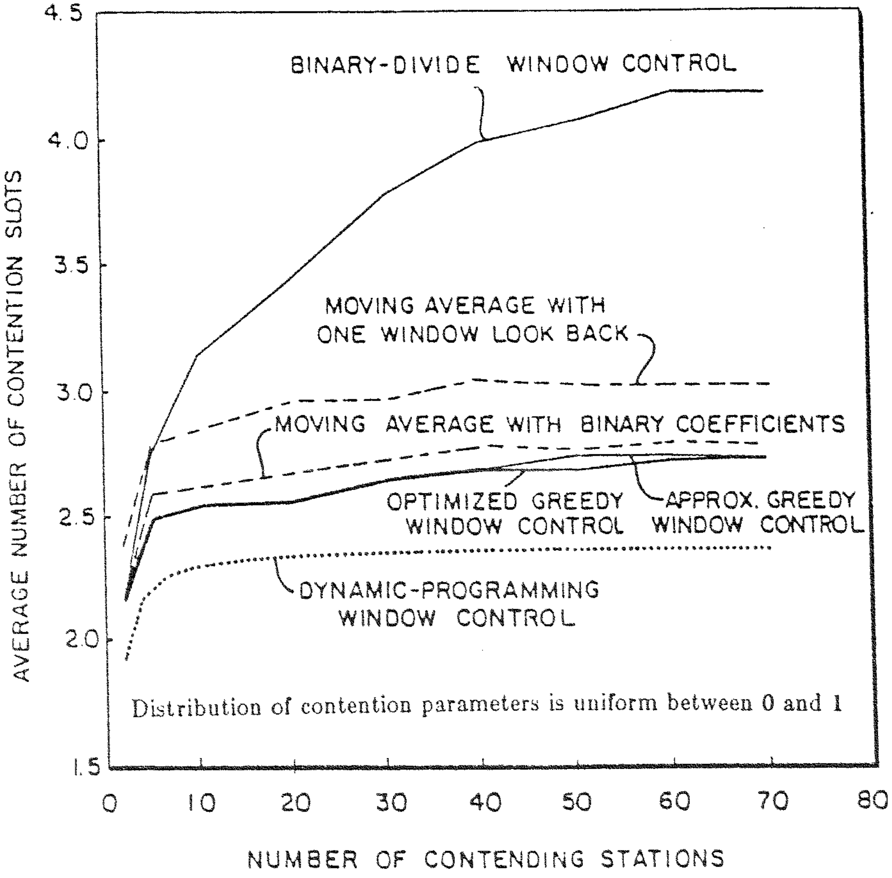


Figure 3.6 The performance of the window protocol with different window controls and load-estimation methods

3.5.2 Dynamic-Programming Window Control

The dynamic-programming window control can be used to minimize the expected total number of contention slots before a successful transmission is possible. The following definitions are first defined:

N(a,b,n): the minimum expected number of contention slots to resolve contention given that there are n contention parameters in the (a,U and collision occurs in the current window (a,b];

g(w,n,a,b): probability of *success* in the next contention slot if a window o (a,w], a<w<b, is used;

ℓ(w,n,a,b): probability of *collision* in the next contention slot if a window c (a,w], a<w<b, is used;

r(w,n,a,b): probability of *no transmission* in the next contention slot if a window of (a,w], a<w<b, is used.

It follows directly form the above definitions that:

$$\ell(w,n,a,b) + g(w,n,a,b) + r(w,n,a,b) = 1 \qquad (3.1$$

The problem here is to find a w that minimizes the expected number future contentions should collision be detected in the current contentio According to the above definitions, this problem can be formulated recursive as:

$$N(a,b,n) = \min_{a<w<b} \left\{ 1 + 0 \cdot g(w,n,a,b) + N(a,w,n) \cdot \ell(w,n,a,b) + N(w,b,n) \cdot r(w,n,a,b) \right\} \qquad (3.1$$

The probabilities g(w,a,b), ℓ(w,a,b), and r(w,a,b) can be derived from the distributions of the contention parameters and the state of contention. When transmission is unsuccessful, it is always possible to identify a window (a,b] such that at least two of the x_i's lie in (a,b] and no x_i is smaller than a. This condition is designated as event A.

A = {at least two x_i's are in (a,b], given that all x_i's are in (a,U]}.

Suppose the window is reduced to (a,w], a<w<b, in the next slot, three mutually exclusive events corresponding to the three possible outcomes can be identified in the next slot:

B = {exactly one of the x_i's is in (a,w], given that all x_i's are in (a,U]};

C = {no x_i is in (a,w], given that all x_i's are in (a,U]};

D = {more than one x_i is in (a,w], given that all x_i's are in (a,U]}.

From these sets of events, the probabilities can be expressed as:

$$g(w,a,b) = \Pr\{B|A\} = \frac{\Pr\{A \cap B\}}{\Pr\{A\}} \qquad (3.16)$$

$$r(w,a,b) = \Pr\{C|A\} = \frac{\Pr\{A \cap C\}}{\Pr\{A\}} \qquad (3.17)$$

The set A∩B represents the event that exactly one of the x_i's is in (a,w], at least one x_i is in (w,b], and all others are in (w,U]. The set A∩C represents the event that at least two x_i's are in (w,b], given that all x_i's are in (w,U].

Let $F_i(x)$ be the distribution that governs the generation of x_i, $1 \le i \le N$, and n be the number of contending stations, then event A occurs with probability:

$$\Pr(A) = \frac{\prod_{i=1}^{n}[1-F_i(a)] - \sum_{i=1}^{n}\left\{[F_i(b) - F_i(a)]\prod_{\substack{j=1\\j\neq i}}^{n}[1-F_j(b)]\right\} - \prod_{i=1}^{n}[1-F_i(b)]}{\prod_{i=1}^{n}[1-F_i(a)]} \quad (3.$$

The first and last terms in the numerator of Eq. 3.18 indicate the probabilit that all x_i's are greater than a and b, respectively. The second term is the p bability that exactly one of the x_i's is in the window (a,b]. Similarly,

$$g(w,a,b) = \frac{\sum_{i=1}^{n}[F_i(w)-F_i(a)] * \left\{\prod_{\substack{j=1\\j\neq i}}^{n}[1-F_j(w)] - \prod_{\substack{j=1\\j\neq i}}^{n}[1-F_j(b)]\right\}}{\Pr(A)\prod_{i=1}^{n}[1-F_i(a)]} \quad (3.$$

$$r(w,a,b) = \frac{\prod_{i=1}^{n}[1-F_i(w)] - \sum_{i=1}^{n}\left[[F_i(b)-F_i(w)]\prod_{\substack{j=1\\j\neq i}}^{n}[1-F_j(b)]\right] - \prod_{i=1}^{n}[1-F_i(b)]}{\Pr(A)\prod_{i=1}^{n}[1-F_i(a)]} \quad (3.$$

It follows that an optimal window can be derived in each step of the co tention process once the channel load and the distribution of the contenti parameters are known. However, the dynamic-programming formulation continuous and requires infinite levels of recursion. Boundary conditions mu be set to terminate the evaluations after a reasonable number of levels encountered. In practice, the x_i's may represent indistinguishable physi measures when their difference is less than δ. It is assumed that when the w dow size is small than δ, the probability that two stations have generat parameters in this interval is so remote that contention can always be resolved in one step. The following boundary condition is included:

$$N(a,b) = 1 \qquad \text{for all } (b-a) < \delta$$

The value of δ was set to $\frac{1}{10 \times n}$, where n is the number of contending stations, in our evaluations for continuous distributions and to one for discrete distributions. The results of evaluation are plotted in Figure 3.6, which shows that the average number of contention slots is bounded by 2.4, independent of the number of contending stations.

Simulations have also been conducted in which the binary-divide rule is used whenever the window size is less than δ. The results obtained are also summarized in Table 3.1, which also show that the number of contention slots is independent of the number of contending stations. The numerical evaluations using Eq. 3.5 are also shown in this table for comparisons. Note that recursion in Eq. 3.15 stops whenever the window size is small than δ, but that the binary-divide rule is applied in the simulations. Hence the performance as obtained by simulations is slightly worse. Nevertheless, the truncation does not affect the performance significantly.

Another set of simulation results with different truncations are shown in Table 3.2. The truncation interval δ were set to $\frac{1}{r \times n}$, where n is the number of contending stations and r is a parameter for varying the size of truncation intervals. The results of numerical evaluation shown in the first column indicate that truncation has little effect on the performance when r is reasonably large. The second column of the table shows the simulation results obtained by converting the dynamic-programming window control to binary-divide window

Table 3.1 Performance of the window protocol with dynamic-programming window control (r=10)

Binary-divide window control is used in the simulations when the window size is small than δ

Performance of The Window Protocol with Dynamic-Programming Window Control		
n	Numerical Evaluations	Simulations
5	2.21	2.33
10	2.29	2.42
15	2.32	2.52
20	2.33	2.43
25	2.34	2.48
30	2.34	2.56
35	2.35	2.57
40	2.35	2.48

Table 3.2 The effect of truncation on the performance of the window protocol with dynamic-programming window control (n=20)

Binary-divide window control is used in the simulations when the window size is smaller then δ

The Effect of Truncation on the Performance of the Window Protocol with Dynamic-Programming Window Control		
r	Numerical Evaluations	Simulations
1	1.54	2.48
2	1.96	2.49
3	2.12	2.49
4	2.19	2.52
5	2.24	2.48
6	2.27	2.56
7	2.29	2.52
8	2.30	2.50
9	2.32	2.45
10	2.33	2.47
11	2.34	2.50
12	2.35	2.55
13	2.35	2.47
14	2.35	2.48
15	2.36	2.42
16	2.36	2.52
17	2.36	2.50
18	2.37	2.51
19	2.37	2.48
20	2.37	2.45

control after the window is smaller than truncation interval δ. The resul indicate that the performance of window protocol is independent of trunc tions. This behavior also reveals that the binary-divide window control is efficient as the dynamic-programming window control after the first step contention. In other words, window control for the first step of contention crucial to the performance of the window protocol.

The results obtained agree with the results of Arrow et al., who had st died a similar problem in economic theory [ARR81]. In their study, the numb of contending stations in a collided window was assumed to be exactly know The problem can be formulated in a finite recursion, and has asymptotic ave age bound of 2.4 steps as obtained by numerical evaluations. We hav obtained comparable results when only tenary information on collision is ava able and the infinite dynamic programming tree is truncated. These resul demonstrate that information on the number of contending stations in a co lided window is not very useful in improving the average performance.

The performance of dynamic-programming window control is much bett then the Binary-Exponential-Backoff algorithm of Ethernet [SHO82] as show by the comparisons of the simulation results depicted in Figure 3.7. The con parison suggests that the performance of Ethernet may be improved by inclu ing a load estimation mechanism into the current industrial standard.

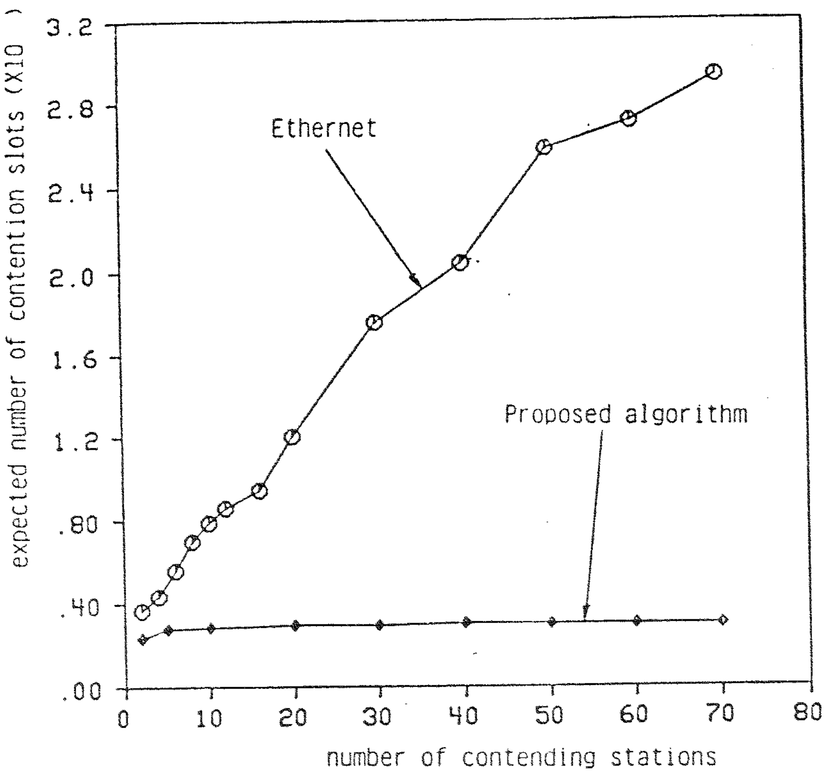


Figure 3.7 Comparison of Ethernet Binary Exponential Backoff algorithm with dynamic-programming window control

3.5.3 Optimal Greedy Window Control

The size of the window will affect the probability that the minimum i successfully isolated. An optimal greedy-window control is one that finds a window to maximize the probability of success, g(w,a,b), in each step of contention. The window-control scheme based on this algorithm is an optima greedy scheme. We have developed an efficient way of finding the window tha maximizes g(w,a,b). When the contention parameters have identical continuous distributions, g(w,a,b) can be expressed in a simple form as:

$$g(w,a,b) = K\,[F(w)-F(a)]\left\{[1-F(w)]^{n-1}-[1-F(b)]^{n-1}\right\} \qquad (3.21$$

where $K = n(1-F(a))^n/P\{A\}$. It can be shown that Eq. 3.21 is unimoda between a and b, so a maximum exists in the window (a,b]. To find th optimal value of w, we set $\frac{d}{dw}[g(w,a,b)] = 0$ and solve for w. This derivatio leads to the following equation if $f(w) \neq 0$:

$$[1-F(w)]^{n-1} - [1-F(b)]^{n-1} = (n-1)[F(w)-F(a)][1-F(w)]^{n-2} \qquad (3.2$$

Let $z = 1-F(w)$, Eq. 3.22 becomes:

$$z^{n-1} - \frac{(n-1)[1-F(a)]z^{n-2}}{n} - \frac{[1-F(b)]^{n-1}}{n} = 0 \qquad (3.2$$

It can be shown that a real root of Eq. 3.23 exists and satisfies the inequali $(1-F(b)) < z_o < (1-F(a))$. There is no closed-form solution to Eq. 3.23, and has to be obtained numerically. Once z_o is obtained, the upper boundary the window, w_o, can be computed directly from z_o as follows:

$$w_o = F^{-1}(1-z_o) \tag{3.24}$$

The performance of the greedy window control scheme is analyzed as follows. Again, we evaluate the average number of contention steps it takes before the minimum is identified. The key issue in the analysis is to show that the lower bound of the maximum success probability, $g(w_o,n,a,b)$ is equal to e^{-1}.

Lemma 1 : The following two relations hold if the contending parameters are generated from a uniform distribution in the interval (0,1):

$$g(w_{n_1},n_1,0,1) \geq g(w_{n_2},n_2,0,1) \qquad \text{if } n_1 < n_2, \tag{3.25}$$

and

$$\lim_{n\to\infty} g(w_{n,}n,0,1) = e^{-1} \tag{3.26}$$

<Proof>

From Eq. 3.21, if F(x) is uniformly distributed, then

$$g(w_n,n,0,1) = nw_n(1-w_n)^{n-1} \tag{3.27}$$

The RHS of Eq. 3.27 is maximized at

$$w_n(0,1) = \frac{1}{n} \tag{3.28}$$

Substituting into Eq. 3.27 yields,

$$g(w_n,n,0,1) = \left(\frac{1}{n}\right)^{n-1} \tag{3.29}$$

It is easy to verify that if $n_1 \leq n_2$, then

$$\left(\frac{1}{n_1}\right)^{n_1-1} \geq \left(\frac{1}{n_2}\right)^{n_2-1} \qquad (3.3$$

Hence, Eq. 3.25 holds. It also follows from Eq. 3.29 that

$$\lim_{n\to\infty} g(w_n,n,0,1) = \lim_{n\to\infty}\left(\frac{1}{n}\right)^{n-1} = e^{-1} \qquad (3.3$$

□

Lemma 2 : If $g(w,n,0,b)$ is maximized at w_n, then $w_n < \frac{1}{n}$.

<Proof>

It can be verified that $g(w,n,0,b)$ is unimodal in the interval $(0,b)$.

Furthermore,

$$\frac{\partial}{\partial w}g\left(\frac{1}{n},n,0,b\right) = \left(\frac{1}{n}\right)^{n-1} -(1-b)^{n-1}-\frac{(n-1)}{n}\left(\frac{1}{n}\right)^{n-2} < 0 \qquad (3.$$

It follows that $\frac{1}{n}$ is on the falling edge of the curve, and hence is grea

than w_n. □

Next, we prove that $g(w_n,n,0,b)$ is a decreasing function of b.

Lemma 3 :

$$\frac{\partial}{\partial b}\, g(w_n,n,0,b) \leq 0$$

<Proof>

Without loss of generality, let F(x) be a uniform distribution. Then, Eq. 3.21 may be rewritten as

$$g(w,n,0,b) = \frac{nw[(1-w)^{n-1} - (1-b)^{n-1}]}{1-(1-b)^{n}-nb(1-b)^{n-1}} \qquad (3.33)$$

Since g(w,n,0,b) is maximized at w_n, then by definition, $\frac{\partial}{\partial w}g(w_n,n,0,b) = 0$. This leads to

$$(1-w_n)^{n-1} - (1-b)^{n-1} = (n-1)w_n(1-w_n)^{n-2} \qquad (3.34)$$

By combining Eq (3.33) and (3.34) yields :

$$g(w_n,n,0,b) = \frac{n(n-1)w_n^2(1-w_n)^{n-2}}{1-(1-b)^{n}-nb(1-b)^{n-1}} \qquad (3.35)$$

Taking the logarithmic function of the above equation and obtaining the derivative of the resulting equation yields:

$$\frac{\partial}{\partial b}\log g(w_n,n,0,b) = \frac{2}{w_n}\frac{\partial w_n}{\partial b} - \frac{n-2}{1-w_n}\frac{\partial w_n}{\partial b} \qquad (3.36)$$

$$- \frac{n(n-1)b(1-b)^{n-2}}{1-(1-b)^{n}-nb(1-b)^{n-1}}$$

The derivatives of Eq. 3.34 with respect to b is:

$$\frac{\partial w_n}{\partial b} = \frac{(1-b)^{n-2}}{(2(1-w_n)-(n-2)w_n)(1-w_n)^{n-3}} \qquad (3.37)$$

Substituting Eq. 3.37 into Eq. 3.36, yields

$$\frac{\partial}{\partial b}\log g(w_n,n,0,b) = (1-b)^{n-2}[1-(1-b)^{n} - nb(1-w_n)^{n-1}] \qquad (3.38)$$

Since $(1-b)^{n} \geq 0$, the RHS of Eq. 3.38 is less than 0 if

$$1-(1-b)^{n} - nb(1-w_n)^{n-1} \leq 0 \qquad (3.39$$

For n=2, it can be shown that w_n is equal to $\frac{b}{2}$. It is also easy to verif that equality of Eq. 3.39 holds when n=2. We have to show that the LH of Eq. 3.39 is a decreasing function of n. From Lemma 2, $w_n \leq \frac{1}{n}$, So

$$e^{-1} \leq \left(1-\frac{1}{n}\right)^{n-1} \leq (1-w_n)^{n-1} \qquad (3.40$$

Combining the LHS of Eq. 3.39 and Eq. 3.40 yields

$$1-(1-b)^{n} - nb(1-w_n)^{n-1} \leq 1-(1-b)^{n} - nbe^{-1} \leq 0 \qquad (3.41$$

Thus, inequality Eq. 3.39 holds. and hence, $\frac{\partial}{\partial b}\log g(w_n,n,0,b) \leq 0$, an g($w_n$,n,0,b) is a decreasing function of b. □

Theorem 3.1 : The average number of contentions is less than 2.7

<Proof>

Let g_i be the maximum success probability of the i-th contention ste Then the average number of contention steps can be expressed as:

$$C = \sum_{i=1}^{\infty} i \times g_i \prod_{j=1}^{i-1}(1-g_j) \qquad (3.4$$

The success probability, $g(w_\infty,\infty,0,1)$, is equal to $\frac{1}{e}$. By a linear transfo mation, a window of any lower bound may be transformed into a win starting from 0. From this result and Lemmas 1 and 3, we have $g_i \geq \frac{1}{e}$, for $i \geq 1$. Hence,

$$C \leq \sum_{i=1}^{\infty} i \times \frac{1}{e} \prod_{j=1}^{i-1} (1-\frac{1}{e}) \leq e \qquad (3.43)$$

□

The performance of the optimal greedy control scheme is suboptimal and approaches 2.7 contention slots for each successful transmission (see Figure. 3.6).

3.5.4 Approximate Greedy Window Control

The approximate greedy window control is similar to the optimal greedy window control except that an approximate optimization to the equation on success probability (Eq. 3.21) is used. Eq. 3.21 may be rewritten as

$$g(w,a,b) = K\,[F(w)-F(a)]\,[F(b)-F(w)] * [1-F(w)]^{n-2} \sum_{i=0}^{n-2} v^i \qquad (3.44)$$

where $v = [1-F(b)]/[1-F(w)]$. A function with a similar structure, $\hat{g}(w,a,b)$, can be obtained by substituting the term $\left[\sum_{i=0}^{n-2} v^i\right]$ by $(n-1)$. That is,

$$\hat{g}(w,a,b) = K'\,[F(w)-F(a)]\,[F(b)-F(w)]\,[1-F(w)]^{n-2} \qquad (3.45)$$

where $K' = (n-1)K$. $\hat{g}(w,a,b)$ is maximum at a position very close to that of $g(w,a,b)$ and can be obtained by solving $\frac{d}{dw}[\log \hat{g}(w,a,b)] = 0$ (see Figure 3.8). We obtain:

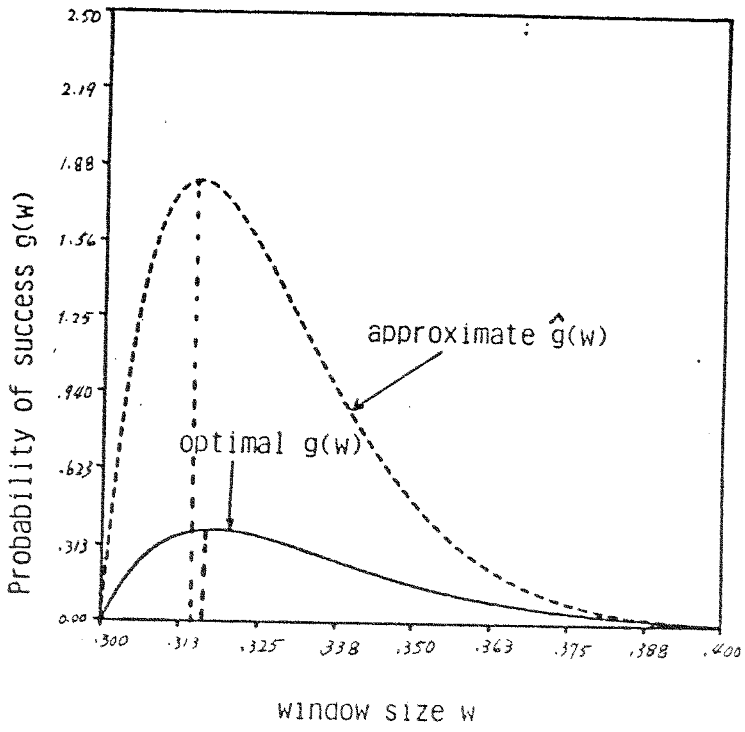


Figure 3.8 Illustration of the maximum probability of successful conten and the maximum of the approximation function ĝ (upper bo = 0.4, lower bound = 0.3)

$$\frac{f(w)}{F(w)-F(a)} + \frac{f(w)}{F(w)-F(b)} + \frac{(n-2)f(w)}{F(w)-1} = 0 \qquad (3.46)$$

or equivalently,

$$[F(w)]^2 + C[F(w)] + D = 0, \qquad (3.47)$$

where

$$C = -\frac{(n-1)[F(a)+F(b)]+2}{n}$$

$$D = \frac{F(a)+F(b)+(n-2)F(a)F(b)}{n}$$

A solution to Eq. 3.47 in the window (F(a),F(b)) is given by:

$$F(w_a) = \frac{-C-\sqrt{C^2-4D}}{2} \qquad (3.48)$$

The approximate window, w_a, as calculated from Eq. 3.48 gives a performance that is nearly as good as that of the optimal greedy scheme (Figure. 3.6). It is worth to note that a binary-divide window control scheme is derived from the optimal greedy window-control scheme by setting the number of contenders, n, to be two. When n=2, both Eq's 8 and 13 have the same solution, i.e., $F(w_o)$ = [F(a)+F(b)]/2. If F(y) is uniformly distributed over (0,1], then w_o = (a+b)/2. The binary-divide control rule can be used as a heuristic for window control with general distribution functions. It can be interpreted as one which assumes that there are two contending processors. As a result, it performs satisfactory when the channel is lightly loaded and degrades to have an $O(\log_2 n)$ performance when the channel load is heavy.

3.6 Load Estimations

Before the window-control protocol is carried out, the number of contending processors is estimated from the distribution of the contention parameter and the statistics of previous channel activities. This information is essential i estimating an initial window and in controlling the dynamic changes in window sizes in the current contention period. A method based on maximum likelihoo estimation is described here.

3.6.1 Maximum Likelihood Load Estimation

After the t'th contention, the window, (a,w(t)] that successfully isolate th station with the minimum is known to all the processors. A maximum likelihood estimate, $\hat{n}(t)$, of the number of stations participated in the conten tion can be computed from a likelihood function, which is the probability c success that the minimum lies in (a,w]. Assuming that the contention parame ters are uniformly distributed in (0,1), the likelihood function may be derive as:

$$L(n(t),w(t),0) = \Pr(0<Y_1<w<Y_2)$$

$$= \hat{n}(t)w(t)(1-w(t))^{n-1} \qquad (3.4$$

L(n(t),w(t),a) is maximized at

$$\hat{n}(t) = \left\lceil \frac{-1}{\ln(1-w(t))} \right\rceil \qquad 0<w(t)<1 \qquad (3.5$$

The number of contending stations in the (t+1)'th contention can be obtaine by adding to $\hat{n}(t)$ the difference between the possible arrivals and departur after the i'th contention. The average number of contention slots to resolve contentions using the optimal greedy window control with this load estimation method is 3.1 as shown in Figure 3.6.

Since the extremum is readily available when a contention is resolved, it can be "piggybacked" in the packet transmitted. Hence an alternative estimate is based on the density function of this statistic. The conditional density of y_1 is derived as:

$$f_{Y_1}(y \mid 0<Y_1<w<Y_2) = \frac{\int_w^1 f_{Y_1Y_2}(y_1,y_2)dy_2}{\int_a^w \int_w^1 f_{Y_1Y_2}(y_1,y_2)dy_2dy_1} \tag{3.51}$$

Since the contention parameters are independent and uniformly distributed in (0,1),

$$f_{Y_1Y_2}(y_1,y_2) = n(n-1)(1-y_2)^{n-2} \tag{3.52}$$

Substituting Eq. 3.52 into Eq. 3.51 yields:

$$f_{Y_1}(y \mid a<Y_1<w<Y_2) = \frac{1}{w} \tag{3.53}$$

This result shows that the distribution of y_1 is determined once the window (a,w] is known. Therefore, no new information is gained by using the first-order statistic in estimating n.

3.6.2 Improving Load Estimation by ARMA

The accuracy of load estimation can be improved by using information o the previous windows that successfully isolate a single station. Instead of usi the window w(t) alone, a techniques in time-series analysis called Aut Regressive-Moving-Average (ARMA) model can be applied to obtain a estimated window based on all previous windows, w(1), w(2), ... , w(t). A sir ple example is to compute a moving average, w_{mv} using the following formu recursively.

$$w_{mv}(t) = \frac{(w_{mv}(t-1) + w(t))}{2} \tag{3.5}$$

The value of $w_{mv}(t)$ is then used in Eq. 3.49 to estimate the channel load. T performance of using ARMA load estimation is very close to that when t channel load is exactly known (Figure 3.6).

3.7 Estimating the Distribution Function of Contention Parameters

In applications such as load balancing and finding the highest priori class, the distribution functions from which the contention parameters are ge erated are unknown and have to be estimated dynamically. Generally, the d tribution functions are assumed, and parameters of the distribution functio are estimated from the statistic collected. Since information on the distrib tion functions is essential and must be consistent for all sites to optimize t window control, independent monitoring of local information and informati broadcast on the bus may be insufficient and may lead to unstable operations.

The proposed window-control algorithms are quite robust with respect to changes in the distribution functions. Experiments on variations of the parameter of a Poisson Distribution did not lead to any significant degradation in performance. However, there is always a delay between the time that the distribution function is changed and the time that this changes is propagated to all sites. The optimization in the window protocol may be unstable if changes cannot be disseminated in time. In this case, a binary-divide window-control scheme is preferable.

3.8 Concluding Remarks

In this chapter we have shown that a class of resource allocation problems may be reduced to the problem of determining the extremum from a set of physically distributed random numbers. A distributed algorithm that identifies the extremum in a constant average time is proposed. This algorithm incurs no explicit message transfer if implemented on a contention bus with the collision-detection capability. The correspondence between the properties of our design and the proposed methodology is summarized in Table 3.3. A load balancing scheduling scheme utilizing the protocol described in this chapter is illustrated in Appendix A.

Table 3.3 Summary of the proposed resource allocation scheme in singl contention-bus networks (bit-serial bus)

Methodology	*Design*
Optimal Allocation	Request of Highest Priority
	Resources of Highest Preference
Distributed Algorithm	Distributed Minimum-Search
Primitive Operation	Window Search
Implementation	Collision Detection
Results	No Explicit Message Transfer
	2.4 Contention Slots (Optimal)

CHAPTER IV
DISTRIBUTED EXTREMUM SEARCH IN BIT-PARALLEL CONTENTION-BUS NETWORKS

We have studied a distributed window-search algorithm to determine the extremum among a set of physically distributed random numbers in a bit-serial single contention-bus network. In this chapter, we concentrate on the design of the window-search scheme on a bit-parallel bus.

A bit-parallel bus is a common communication network in computer systems, and is composed of multiple parallel links thru which data can be transmitted in parallel (Figure 4.1). It connects registers and arithmetic-logic units (ALU) in central processors (CPU). It is also used to connect CPU and memories or peripherals. There are many proposed bus-arbitration schemes [THU72, HAY79, BAE80, HWA84]. These schemes performs well when number of processors connected to the bus is small. However, their performance may degrade drastically when number of processors is large.

A deterministic Multiaccess Code-Deciphering (MACD) scheme for efficiently scheduling bus access is presented in this chapter. Its scheduling overhead is small and independent of number of processors. Furthermore, it is capable of determining the extremum among a set of distributed random numbers without any explicit massage transfer. Only the algorithm for identifying maximum is discussed here. The algorithm for identifying minimum is similar.

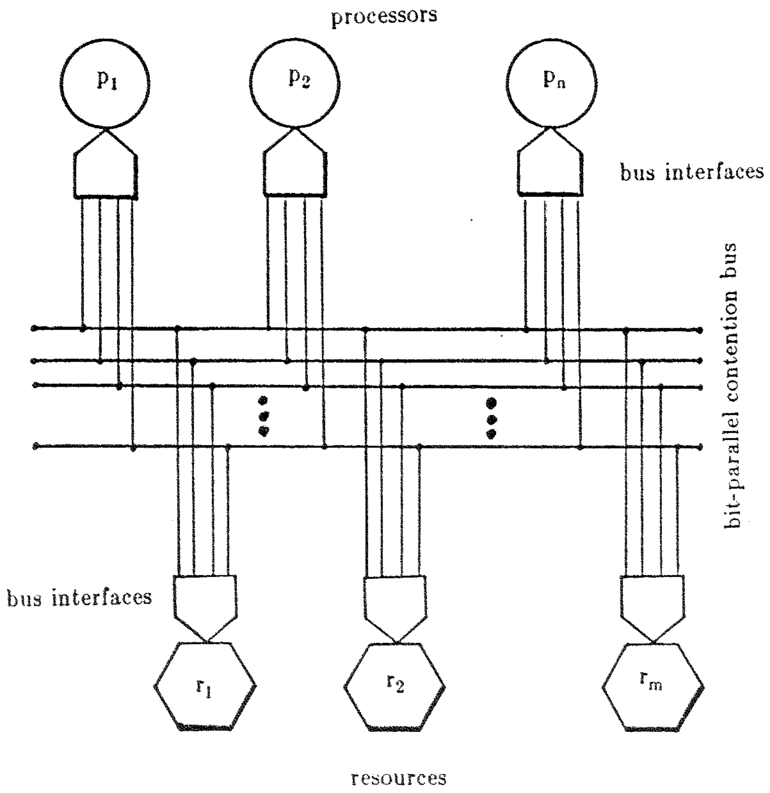


Figure 4.1 A resource sharing computer system connected by a bit-paral contention bus

4.1 Multi-Access Code-Deciphering Contention Resolution

In the bit-serial window-search algorithm discussed in the last chapter, a global window is maintained by all contending processors, each of which generates a contending parameter before contention begins. A contending processor is eliminated from contention if its parameter is outside the window. A distributed window-control rule is applied to expand or to shrink the window in each contention step. As the contending process proceeds, the window size becomes smaller and smaller. Eventually, the processor that has generated the minimum parameter will be isolated in the window. These window-control algorithms are based on information of previous contentions and an estimate of the channel load. On the other hand, a bit-parallel bus is usually used to connect components in close proximity. The data transfer rate is so high that an even faster bus-arbitration scheme is necessary. A similar window-control scheme as used in a serial bus is too complicated to be useful in this environment. The Code-Deciphering technique to be discussed next is a fast and effective scheme that combines window control and collision detection in a simple manner to cope with the problem.

To adapt the window-control scheme to a bit-parallel bus, two mechanisms are needed: a collision-detection mechanism and a window-access mechanism. The collision-detection mechanism can be implemented by the Wired-OR property of the bit-parallel bus. When two or more processors write

simultaneously on the bus, the result is simply the bitwise logical OR of the numbers. By interpreting the result after a write, each processor can determi whether a collision has occurred. To describe the scheme formally, let assume that there are N requesting processors, and each processor writes binary number X_i ($i=1,2,...,N$) to the bus. For example, assume that there a three processors, and that each writes the following numbers, $X_1 = 1001$ $X_2 = 0101_B$, and $X_3 = 0100_B$ on the bus. The bitwise logical OR of the numbers is 1101_B, which is different from X_1, X_2, or X_3. Thus, every process knows that a collision has happened by examining this number instead exchanging messages. The decision to write another number on the bus can determined by interpreting the data read from the bus.

Suppose the X_i's are chosen from a structured code space S with the fc lowing properties:

(1) $X_i, X_j \in S$, $i \neq j$, are linearly related, i.e., $X_i > X_j$ or $X_i < X_j$ (4.1.

(2) $f(X_1 \oplus X_2 \oplus \cdots \oplus X_N) \leq \max\{X_1, X_2, ..., X_N\}$ (4.1

$X_i \in S$, $N \geq 1$

where $\oplus$ is the bitwise logical OR operator. By reading data on the bus ar applying f, the code-deciphering function, a processor knows the maximu number written on the bus. This information provides a basis for the windov search mechanism to set another window. If the initial window is set such th the maximum value is included in the window, then an optimal detection pr cedure can be designed to isolate exactly one processor eventually.

4.2 Implementation of MACD

Using the code deciphering scheme described above, a bit-parallel window-search scheme can be designed. The network supporting the protocol should have the following components: a synchronous parallel bus for transmitting data and codes, a bus-status control line for indicating the busy status of the bus, and an intelligent processor-bus interface for each processor that is capable of (a) sensing the bus-status, (b) reading data from the bus, (c) writing data to the bus, (d) generating random codes, and (e) deciphering codes read from the bus. The time interval for generating a random number, writing the number to the bus, and deciphering the code read from the bus is called a *contention slot*. Whenever a processor has data ready to transmit, it senses the bus status first. If the bus is in use, then it waits until the bus becomes idle. To contend for the bus, a processor chooses a code randomly from the code space S and writes it to the bus. The resulting code written on the bus is the bitwise logical OR of all the codes written by the contending processors. Each contending processor reads the resulting code written and computes the deciphered code using the code-deciphering function. It compares the deciphered code with the code generated locally. Three outcomes may result:

(1) the locally generated code is equal to the code read;

(2) the locally generated code is not equal to the code read but is equal to the deciphered code; and

(3) the locally generated code is equal to neither the code read nor the deciphered code.

The last outcome implies that this processor has not generated the max
imum code and hence can be eliminated from further contention. The first an
second outcomes imply that this processor has generated the maximum cod
and should be allowed to transmit. However, there may be more than one pro
cessor that has generated the same code. If there is more than one processor i
this set, a ***hidden collision*** is said to occur, and the contention-resolution pro
cess has to be repeated.

There are two ways to detect a hidden collision. First, each processor i
this set generates an n-bit random number and writes it to the bus. T
prevent the possibility of two processors generating the same random numbe
each processor can use a distinct n-bit station-identification code as the rando
number. If the number read from the bus matches with the number writte
then the hidden collision has been resolved. If collision is detected, then th
MACD scheme is repeated. Second, we can assume that the hidden collision
not resolved, and the collision-detection process is repeated. The process has
be repeated a number of times until there is a high confidence that exactly o
processor is isolated. When the probability that a large number of statio
have generated the maximum code is high, the second method of resolving hi
den collision is better because the MACD process is likely to be repeated, a
the time for propagating the random number in the first method is lost. C
the other hand, if the probability that exactly one station has generated t
maximum code is high, then the first method is better because hidden collisi
can be detected efficiently. Usually, the second method is used when the co
space **S** is much smaller. As a result, a few additional steps are necessary
achieve a high enough confidence that there is no hidden collision. In th

chapter, we have used the first method of resolving hidden collisions because the number of contending processors is usually relatively small as compared to the bus width. Another method is proposed in Section 4.5 to cope with a large number of contending stations.

For the Code-Deciphering technique to work properly, a code space that satisfies Eq's 4.1a and 4.1b must be constructed, and a code deciphering function must be designed. Two code spaces together with their respective deciphering functions are presented.

4.2.1 Unary Code

Let the code space **S** be taken from the set $\{0^a 1 0^b \mid a+b = n-1, a \geq 0, b \geq 0\}$ where 0^k represents a consecutive sequence of k zeroes. Then for any two different elements u and v in **S**, it is easy to verify that the linear ordering property holds in this space. For any n-bit binary number, $X=(x_1 x_2 \ldots x_n)$, we define a deciphering function f on X such that:

$$f(X) = 0^p 1 0^{n-p-1}, \quad \text{if } x_{p+1}=1,\ x_j=0 \text{ for all } 0 \leq j \leq p.$$

We claim that **S** and f as defined above satisfy Eq's 4.1a and 4.1b. To verify this, we consider N codes such that:

$$c_i = 0^{a(i)} 1 0^{n-a(i)-1}, \quad i=1, \ldots, N$$

By the definition of **S**,

$$c_i \in \mathbf{S}, \text{ and}$$

$$\max(c_1, c_2, \ldots, c_N) = 0^m 1 0^{n-m-1}$$

where $m = \min\{a(i) \mid i=1,2,\ldots,N\}$. An overlapped variable $Y=(y_1 y_2 \ldots y_n)$ is defined to be the bitwise logical OR of the c_is, i.e.,

$$(y_1 y_2 \cdots y_n) = c_1 \oplus c_2 \oplus \cdots \oplus c_N.$$

Y as defined retains the following properties:

$$y_{m+1} = 1, \text{ and}$$

$$y_k = 0 \quad \text{for } 1 \leq k \leq m.$$

By the definition of the deciphering function f,

$$f(Y) = 0^m 1 0^{n-m-1} \text{ or}$$

$$f(c_1 \oplus c_2 \oplus \cdots \oplus c_N) = \max(c_1, c_2, \ldots, c_N).$$

It will be shown in Section 4.4 that $\frac{1}{W}$ (W is the bus width) stations remain after each contention slot.

4.2.2 Binary Code

If binary codes are used, Eq. 4.1a is still satisfied. A new code-deciphering function has to be designed. By detecting the most significant bit that is mismatched among the codes generated by the contending processors, half of stations on the average can be eliminated in each contention slot. This is not as efficient as unary-code representations. A similar scheme can be found in [MOK79].

4.3 Determining the Maximum Priority Using MACD

4.3.1 Code-Space Partitioning

To determine the maximum priority level, the code space of the original MACD scheme is partitioned into subspaces such that each subspace corresponds to a priority level. The partitions should satisfy the following condition:

$$\text{If } X \in S_i,\ Y \in S_j, \text{ and } i > j, \text{ then } X > Y$$

where S_i and S_j are subspaces corresponding to priority levels i and j respectively. Using this partitioning, priority levels are encoded into the contending codes, and the deciphering function proposed in Section 4.2.1 can identify the highest priority level.

4.3.2 Parameter-Space Partitioning

The partitioning of the code space is practical when the number of priority levels is relatively small as compared to the size of the code space. When the number of priority levels is large, a parameter-space partitioning approach can be used. A strictly increasing function which maps priority levels onto the code space is defined in each contention slot. The mapping is done in such a way that the minimum number of priority levels is assigned the same code. In a contention slot, every contending processor writes the code associated with its priority to the bus and deciphers the number read from the bus. The subrange that maps to the maximum code is identified. A new function that maps this

subrange into the whole code space is defined, and the process is repeated unti the subrange contains only one priority level. This method is illustrated in Fig ure 4.2. In the first step, a set of the priority levels, including level x_1, ar mapped to code C_2. The first contention step shows that C_2 is the maximur code. Accordingly, all the priority levels that maps to C_2 are allowed to con tend in step 2. The second contention identifies C_1 as the maximum code. I the last step, C_3 is the maximum code and level x_1 is the only priority leve that mapped to this code, so priority level x_1 is the maximum priority level.

4.4 Evaluation of MACD Contention Resolution Scheme

The analysis and simulation results are shown in this section. The tim complexity of contention resolution can be measured by the mean number c contention slots expended before the maximum can be identified. To analyz this complexity, let N be the number of contending processors at the beginnin of a contention period and K be the size of the code space, which is equal t the bus width W. Assuming that priority levels are generated randomly, the a processor generates a given code c_i $(i = 1, 2, ..., N)$ with probability 1/V Designate the maximum of N such c_i's as c_m, the m-th code in the code spac i.e., $c_m = \max\{c_i | \ i = 1, 2, ..., N\}$. If exactly one processor generates code c and other processors generate codes less than c_m, then the contention resolved. The probability for this event to occur is:

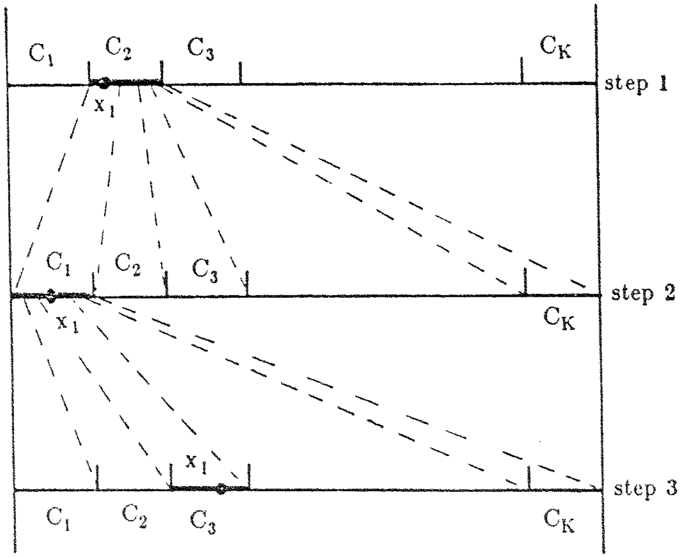


Figure 4.2 Parameter-space partitioning for determining the maximum using the MACD contention-resolution scheme

$$q(m|N,K=W) = N\left(\frac{1}{K}\right)\left(\frac{m-1}{K}\right)^{N-1}$$

Since the maximum code can acquire one of the W possible values, and th
W events are mutually exclusive, the probability that contention is resolvec
one step is $P_{K,W,N}$ where K=W is:

$$P_{K,W,N} = \sum_{m=1}^{K} q(m|N,K=W)$$

$$= \sum_{m=1}^{K} N\left(\frac{1}{K}\right)\left(\frac{m-1}{K}\right)^{N-1}$$

$$= \frac{N}{K^N}\sum_{u=1}^{K-1} u^{N-1}$$

In Figure 4.3, $P_{K,W,N}$ is plotted against N/W. It is observed that the proba
ity of success in one attempt is higher if the code space (equal to the
width) is larger and the number of contending processors is kept constant.
observed that $P_{K,W,N}$ is a strictly decreasing function of N and decreases
zero when N is large. This means that MACD is unable to resolve content
in one step when the load is extremely heavy. However, most of the conte
ing processors are eliminated in one attempt. The number of survivor
reduced significantly as contention proceeds, and the probability of succes
increased consequently. The following analysis demonstrates this phenomen

Given that the maximum of codes generated by the contending proces
is c_m, the m-th code in the code space. Define indicator variables X_i, i=1, ...

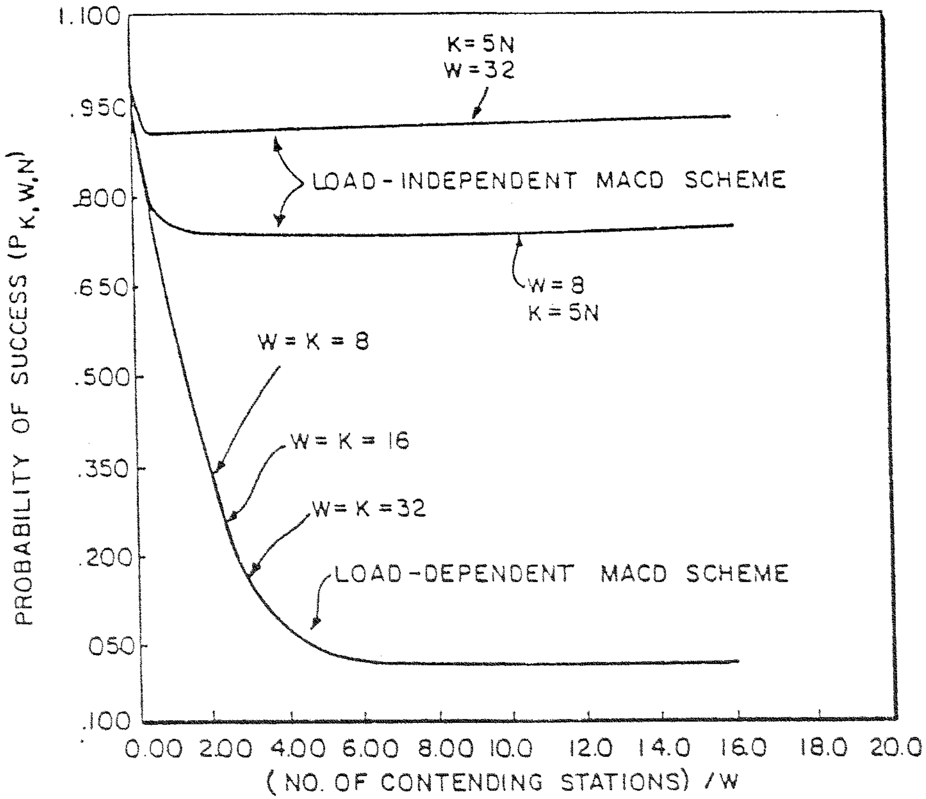


Figure 4.3 The probability that the maximum is determined by a MACD scheme in one step of contention

$$X_i = \begin{cases} 1 & \text{with probability } 1/m \\ 0 & \text{with probability } 1 - 1/m \end{cases}$$

Let

$$.Z = \sum_{i=1}^{N} X_i.$$

The random variable Z indicates the number of processors that generate c_m the contention. These processors are allowed to contend in the following step The expected value of Z given m, N, and W, E(Z|m,N,W), represents the ave age number of surviving processors. It is easy to show that:

$$E(Z|\,m,N,W{=}K) = \frac{N}{m}. \qquad (4.$$

Furthermore, the probability that the current maximum code with N conten ing stations and a bus width of W is c_m can be expressed as:

$$p(m|\,N,W{=}K) = \left(\frac{m}{K}\right)^N - \left(\frac{m-1}{K}\right)^{N.} \qquad (4.$$

The expected number of processors that would survive a contention is:

$$E(Z|\,N,W{=}K) = \sum_{m=1}^{K} E(Z|\,m,N,W{=}K)\,p(m|\,N,W{=}K)$$

$$= \sum_{m=1}^{K} \left(\frac{N}{m}\right)\left[\left(\frac{m}{K}\right)^N - \left(\frac{m-1}{K}\right)^N\right]$$

$$= \left[\left(\frac{N}{1} - \frac{N}{2}\right)\left(\frac{1}{K}\right)^N\right] + \left[\left(\frac{N}{2} - \frac{N}{3}\right)\left(\frac{2}{K}\right)^N\right]$$

$$+ \cdots + \left[\left(\frac{N}{K-1} - \frac{N}{K}\right)\left(\frac{K-1}{K}\right)^N\right] + \left[\frac{N}{K}\left(\frac{K}{K}\right)^N\right]$$

$$= \frac{N}{K^N}\left\{\frac{1^{N-1}}{2} + \frac{2^{N-1}}{3} + \cdots + \frac{(K-1)^{N-1}}{K}\right\} + \frac{N}{W}$$

$$\leq \frac{N}{K^N}\left\{K\cdot\frac{(K-1)^{N-1}}{K}\right\} + \frac{N}{K}$$

$$\leq \frac{N}{K}\left(\frac{K-1}{K}\right)^{N-1} + \frac{N}{K}$$

$$\leq \frac{2N}{K} \qquad (4.6)$$

The ratio $\gamma = \frac{E(Z|N,W=K)}{N} \leq \frac{2}{K}$ is a measure of the average fraction of contending processors that can survive a contention. Let N_t $(t=0,1,...)$ be the expected number of contending processors in step t. By Eq. 4.6, we have

$$N_t \leq \left(\frac{2}{K}\right)^t N_0 \qquad t \geq 0.$$

Therefore,

$$N_t \rightarrow 1 \text{ as } t \rightarrow \log_{k/2} N_0. \qquad (4.7)$$

As shown in Figure 4.3, we can see that $P_{K,W,N} \rightarrow 1$ as $N<W$, and $P_{K,W,N} \rightarrow 0$ as $N>>W$. This fact reveals that the contention process of MACD can approximately be divided into two stages. The effect of the first stage, i.e., when $N_t > W$, is in reducing the number of contending processors. When the process enters the second stage, i.e., $N_t \leq W$, contention can be resolved in

about one step. The overall contention process will stop within an average o $\log_{W/2} N_0$ steps. Figure 4.4 shows the simulation results which confirm ou analysis. The number of contention slots shown include the additional slot required for resolving hidden collisions. MACD performs better when the bu width is large.

4.5 Adaptive MACD Contention Resolution Scheme

As shown in Eq. 4.7 and Figure 4.4, the performance of the MACD schen proposed in the last section is load-dependent and performs well when the bu width is large and the number of contending processors is small. Since th number of contention slots grows logarithmically with the number of contend ing processors, the scheme is inefficient when the number of contenting proce sors is large or the bus width is small.

The cause for the load dependency is due to the fixed code space. In ord to reduce the number of processors contending in a slot, the code space can b designed in such a way that it is a function of the number of contending pr cessors and the bus width. By choosing the size of the code space that kee the number of processors contending in a slot to be a relatively small consta as compared to the bus width, contention can be resolved in a time that load-independent. This result is similar to the window-search schemes f carrier-sense-multiple-access networks described in last chapter.

The solution depends on choosing the size of the code space and estima ing the number of contending processors. Suppose N can be estimat

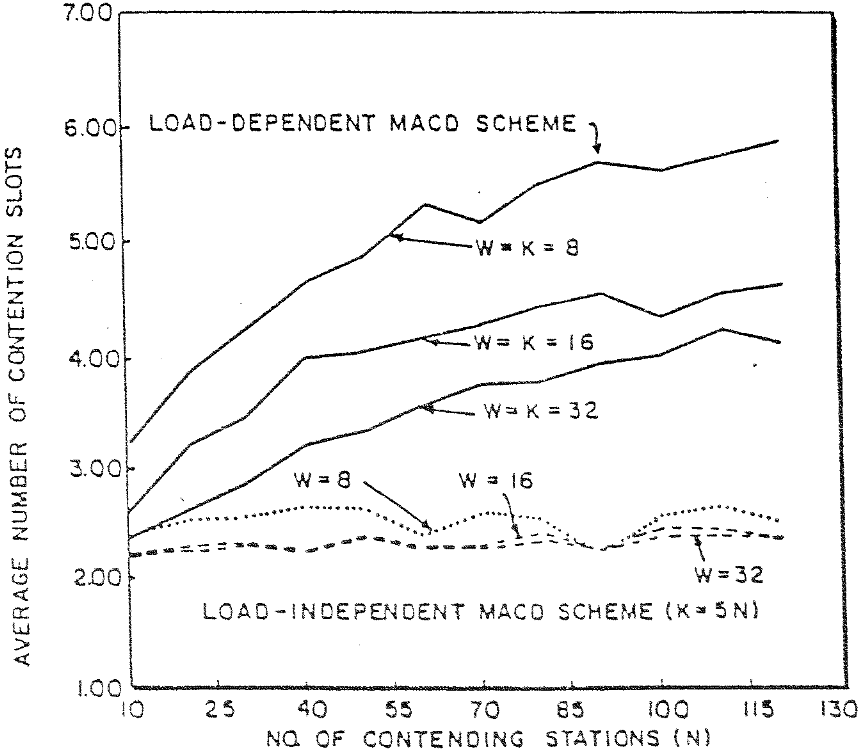


Figure 4.4 Average number of contention slots for resolving conflicts of bus requests using MACD schemes

accurately, and K/N = r. The probability that contention is resolved in o

step (refer to Eq. 4.3) is:

$$P_{K,N,W} = \sum_{m=K-w+1}^{K} q\left(\frac{K}{r},K,W\right)$$

$$= \frac{N}{(r \times N)^N} \sum_{u=K-W}^{K-1} u^{N-1} \qquad (4.$$

where q(m|N={K over r},K,W) is defined in Eq. 4.2. The value of $P_{K,N,W}$ again plotted in Figure 4.4. It is seen that the success probability is higher a load-independent as a result of the increase in the code-space size.

The expected number of processors that would survive a contention c also be derived similarly. In this case, the number of surviving processors is if no station contends in the slot. That is, Eq. 4.4 becomes:

$$E\left(\frac{K}{r},W\right) = \begin{cases} \frac{N}{m} & K \leq m \leq K-W+1 \\ N & 1 \leq m \leq K-W \end{cases} \qquad (4$$

The definition of p(m|N,W) in Eq. 4.5 remains true. The expected number surviving processors in one contention is:

$$E\left(\frac{K}{r},W\right) = \sum_{m=1}^{K} E\left(\frac{K}{r},W\right) \times p\left(\frac{K}{r},W\right)$$

$$= \sum_{m=K-W+1}^{K} \frac{N}{m} \cdot p\left(\frac{K}{r},W\right) + \sum_{m=1}^{K-W} N \cdot p\left(\frac{K}{r},W\right)$$

$$\leq \frac{2N}{K} + \left[\left(\frac{K-W}{K} \right)^N - \left(\frac{K-W-1}{K} \right)^N \right] (K-W)N \qquad (4.10)$$

Since K/N=r is a constant, E(Z|N={K over r},W) is a constant independent of load (equals N) if K is large as compared to W.

The correct choice of r is shown in Figure 4.5. There is an optimal choice of r to minimize the number of contention slots. This optimal value depends on W and is load-independent (assuming that N is known). It is approximately 5 for the combinations of W and N tested. Using the optimal value of r, the performance of the load-independent MACD scheme is plotted in Figure 4.4. In generating these results, the size of the code space, K, is chosen to be W if r×N is smaller than W; that is, the scheme proposed in Section 4.4 is used when the load is light. It is observed that the proposed scheme requires a small constant number of slots even when the load is heavy.

The proposed scheme requires N to be known. In general, this is not possible due to the distributed control. One way is to estimate N based on information collected during the contentions. However, this information can indicate that one or more contending processors have generated the same code, but cannot reveal the exact number of contending processors. This probability is small if the number of processors contending in a contention slot is small compared with the bus width. A reasonable estimate of N can be obtained by using the number of bits that are ones in a contention slot, B, as the number of processors contending in this slot. That is,

$$\hat{N} = \frac{B \times K}{W} \qquad (4.11)$$

This will systematically under-estimate the actual value of N, and some

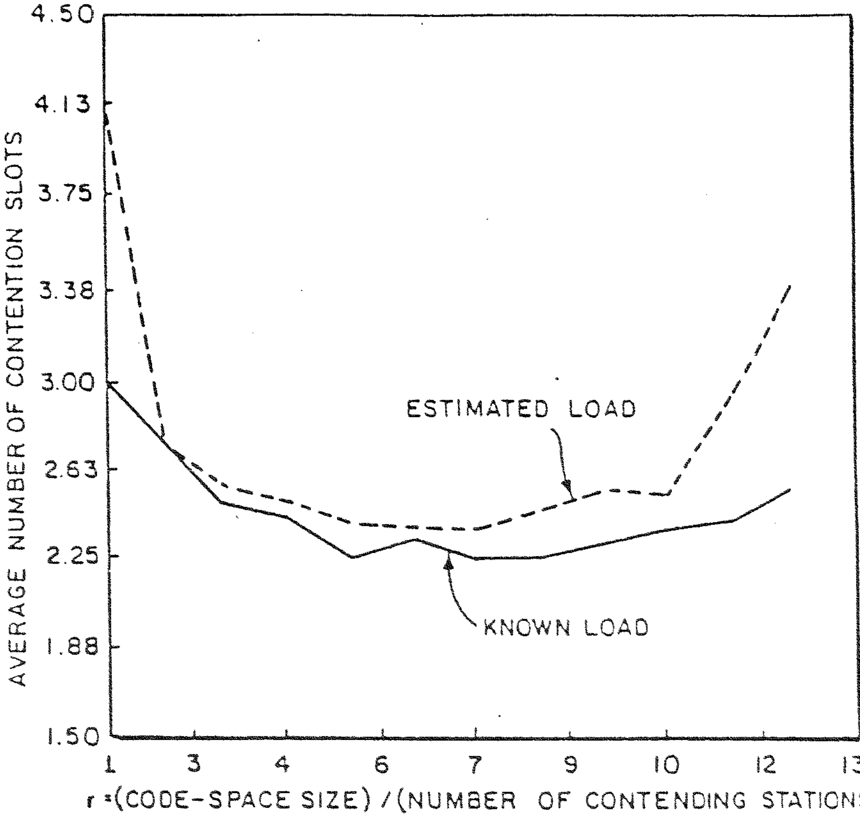


Figure 4.5 The choice of the size of code space

correction to the value of r used should be introduced. In Figure 4.5, the optimal value of r that should be used is slightly different when the estimate in Eq. 4.11 is used. The number of contention slots required is slightly increased when N is estimated (Figure 4.5). A maximum-likelihood estimate of N can also be derived. However, the complexity of such a scheme is high and cannot be efficiently implemented in bit-parallel bus networks. Moreover, choosing a non-optimal value of r does not degrade the performance of the MACD scheme significantly since the flat area of the curve in Figure 4.6 is very wide. The MACD scheme is, therefore robust.

4.6 Concluding Remarks

In this chapter, we have proposed the MACD scheme for a bit-parallel contention bus, and have shown that the MACD scheme is a primitive operation of determining the maximum among a set of distributed random numbers. Thus, the optimal resource allocation algorithms for the class of scheduling discipline characterized in Chapter 3 can be implemented in a system with bit-parallel bus using the MACD scheme. The implementation is similar to that of a system with bit-parallel bus and is summarized in Table 4.1. The MACD scheme is also an efficient bus-arbitration scheme, which is capable of resolving bus contention in a constant time independent of the number of contending processors.

Table 4.1 Summary of the proposed resource allocation scheme in singl contention-bus networks (bit-parallel bus)

Methodology	*Design*
Optimal Allocation	Request of Highest Priority
	Resources of Highest Preference
Distributed Algorithm	Distributed Extremum-Search
Primitive Operation	MACD scheme
Implementation	Code Deciphering
Results	No Explicit Message Transfer
	Constant Number of Contention Slots

CHAPTER V
RESOURCE ALLOCATION IN MULTIPLE CONTENTION-BUS NETWORKS

5.1 Architecture of Multiple Contention-Bus Networks

A multiple contention-bus network provides multiple communication channels between processors and resources (Figure 5.1). These channels may be obtained by modulating a physical bus into multiple channels of different frequencies or simply provided by multiple baseband physical buses. Access to the individual channels follows the same protocol as used in the single contention-bus networks described in Chapter 3. Every processor in the network is capable of transmitting data to every bus or receiving data from them. There is usually a transceiver in each interface to allow parallel transmission of data to the different channels. If all channels share one transmitter, then only one data stream can be transmitted at any time. However, in a contention network, every processor should provide a receiver for each channel to be capable of receiving data from all channels simultaneously. Different configurations of buffer arrangement are shown in Figure 5.2.

A multiple-bus system has the following advantages: (1) The structure of the network is simple and regular. (2) It provides ample bandwidth. (3) It is of highly modularity and easy to expand. (4) It can survive individual bus

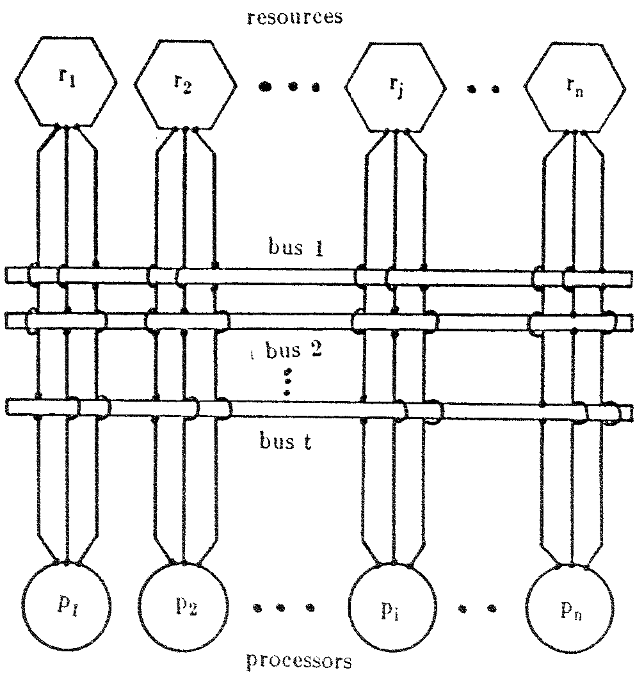
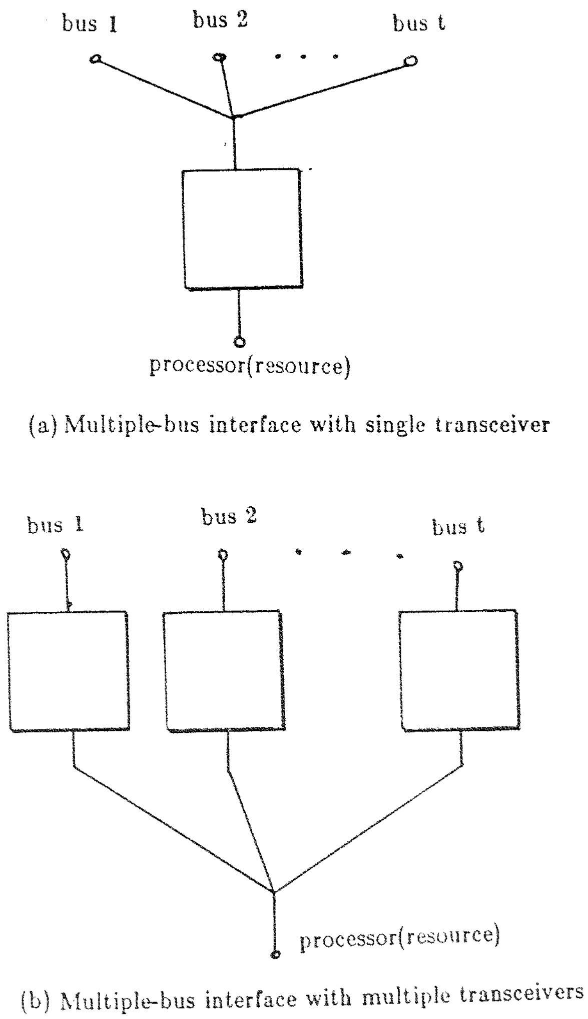


Figure 5.1 A resource sharing computer system connected by multiple buses



(a) Multiple-bus interface with single transceiver

(b) Multiple-bus interface with multiple transceivers

Figure 5.2 Transceiver arrangement in the interface between a processor and multiple buses

failure. (5) It is less expensive than a single bus of the same bandwidth. multiple bus has been proposed for connecting processors and memory modu in multiprocessors [ENS77, LAN83, MAR82b], and channel access meth have studied for multiple CSMA/CD networks [MAR82a].

5.2 Resource Scheduling for Multiple Contention Buses

As described above, a multiple bus network may be expanded from a s gle bus network by adding more communication channels to the system. T resource allocation problem for this type of networks is also an extension of counterpart in single contention-bus networks. Suppose that there are t cha nels, the issue of resource scheduling is to allocate t pairs of processors a resources such that the total cost of allocations is minimized. In general, t total cost can be represented as a function of t (processor priority - resou preference) pairs selected. Using the same conventions as discussed in Chap 3, the problem can be formulated as follows.

$$\min_{(p_i,s_i)\in Q\times R} H((x_{p_1},y_{s_1}),(x_{p_2},y_{s_2}),\ldots,(x_{p_t},y_{s_t})) \qquad (5$$

If requests are independent, then the cost is the summation of the cost of ea individual pair. Hence, Eq. 5.1 can be simplified as:

$$\min_{(p_i,s_i)\in Q\times R} \sum_{i=1}^{t} H(x_{p_i},y_{s_i}) \qquad (5$$

For most scheduling disciplines, the independence relation described in Chap 3 holds.

$$\frac{\partial}{\partial x_{p_i}} H(x_{p_i},y_{s_i}) \leq 0 \tag{5.3}$$

$$\frac{\partial}{\partial y_{s_i}} H(x_{p_i},y_{s_i}) \leq 0$$

This equation implies that requests and resources may be selected separately. However, the question on the method of selecting processors and resources remains to be answered. The following theorem reveals that the t processors of the highest priority and t resources of the highest preference should be selected.

Theorem 5.1: To minimize the cost of resource allocation in multiple-bus networks, if the cost function of allocation satisfies Eq. 5.3, then t processors of the highest priority and t resources of the highest preference should be selected.

<Proof>

The theorem is proved by contradiction. Assume that the mapping T = $\{(p_i,s_i) \mid 1 \leq i \leq t\}$ is optimal but the priority x_{p_i} is not among the t highest ones, i.e., there exists a processor, q, which is not allocated and

$$x_q > x_{p_k} \quad \text{for some k such that } 1 \leq k \leq t \tag{5.4}$$

According to Eq. 5.3, the function H(x,y) is a decreasing function of x. Therefore, replacing Eq 5.4 into Eq 5.2 yields

$$\sum_{j=1,j\neq k}^{t} H(x_{p_j},y_{s_j}) + H(x_q,y_{s_k}) \leq \sum_{j=1}^{t} H(x_{p_j},y_{s_j}) \tag{5.5}$$

This equation indicates that there is another mapping with a smaller cost than that of T. Hence, the mapping T is not optimal, which contradicts the assumption. To obtain the optimal mapping, the processors of the

highest priority must be chosen. With a similar argument, resources (the highest preference should be selected. Thus, the theorem follow

□

For the allocation of single-resource requests, an optimal mapping specified by a set of ordered pairs. The above theorem only characterizes tl domain of this mapping, the pairing among the selected processors and tl selected resources remains to be specified. In other word, a selected resourc has to be assigned to a selected processor after the selections. The procedure (the above scheduling may be summarized as follows:

S1. The t resources of highest priority are selected and their costs are broac cast on the bus.

S2. The t processor of highest preference are selected and their costs ar broadcast on the bus.

S3. The pairing between the selected processors and resources are carried ou at every selected processors concurrently, and each pair of processor an resource are assigned a bus.

S4. The processor allocated with a resource and a bus transmits its requests.

There are t! possible mappings between the selected processors and tl selected resources. It can be shown that finding the best mapping is equivaler to the classical stable marriage problem [MCV71], which can be solved b existing algorithms. In practice, pairing the minimum-cost processor to tl minimum-cost resource sequentially is a good heuristic. Before the optim: mapping can be found, the t processors of the highest priority and the resources of the highest preference have to be identified. This is the problem (

selecting the t smallest numbers from a set of distributed random numbers.

To find the t processors of the highest priority and the t resources of the highest preference, a centralized scheme would require the cost information of all processors and resources to be collected. The collection of these information represents a significant overhead of resource scheduling in a distributed environment. Alternatively, a distributed ordered selection algorithm similar to the distributed minimum-search algorithm of Chapter 3 was developed in this thesis. The algorithm will be discussed in section 5.4. For the class of scheduling disciplines in which no priority and preference are concerned, resource scheduling may be further simplified and integrated with the channel routing schemes. This integration is described in the next section.

5.3 Random Resource Allocation with Distributed Routing

Packet routing that selects a preferred path for transmitting packets is an important problem in computer networks. In a multiple-bus network, packets may be transmitted through any available channel. The routing mechanism may select a channel which is the best (according to some measures). For the scheduling with a random allocation strategy, a request may be treated as a packet and relies on the routing mechanism to be assigned a channel. After a channel is allocated, the request is broadcast. Upon receiving a broadcast request, resources may send an acknowledgement to the source of the request. Again, the routing mechanism will resolve conflicts of acknowledgements generated by the multiple resources. Acknowledgements to a requests are

abandoned if the request has already received one.

The design of routing schemes in a multiple-bus network is much simpl than that of a general network. The routing strategy used has significa effects on the throughput of the network, and hence on the efficiency of tl resource allocations. Conventional routing strategies are investigated belo with the aid of a queueing model. A more efficient implementation is propose later.

5.3.1 A Queueing Network Representation of Packet Routing Scheme

A packet routing scheme in multiple-bus networks may be described terms of a queueing network (Figure 5.3) in which a channel is represented by server. The processors that are contending for a channel are queued to tl corresponding server, and a "random" service discipline is used to model tl random bus-accesses. The length of a queue is referred to be the *channel lo* of its associated channel. A request generator represents a processor. Tl requests are assumed to be generated by a request generator with a Poiss process. The routing decision of a processor is modeled by the branches leadi from the associated request generator to servers.

5.3.2 Probabilistic Routing

Depending on routing strategy, a processor may route its ready request a randomly chosen free channel. Such a scheme is called a *probabilistic rou ing*. Since channels are chosen randomly, a processor may happen to select channel that is heavily loaded. Such an improper routing usually leads to imbalance of workload among channels and results in a poor chann

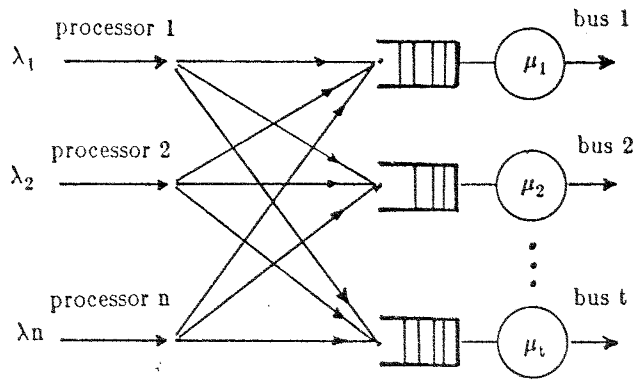


Figure 5.3 A queueing-network representation of the distributed routing scheme in multiple-bus networks

utilization. To improve the performance of probabilistic routing, the branching probabilities may be optimized with respect to the statistic of the network, which include request rates, bus speeds, and data transmission times [CHO79]. The effect of the optimization is on the balance of workload among the channels. Ideally, it would result in a balanced system of which the traffic intensities to all channels are equal.

We have evaluated the performance of probabilistic routing by measuring the percentage of time that the channels are wasted. A channel is wasted when it is idle and there are requests waiting at other channels. Let W be the number of channel being wasted at any instant, then

$$W = min \left\{ \text{(number of idle channels), (number of waiting requests)} \right\}$$

W is a random number with a mean value that can be evaluated accordin to the following assumptions. (1) The network consists of t channels. (2 Requests for transmission are generated independently from each processor an are governed by a Poisson process. (3) The duration of the total of th contention-resolution and packet-transmission times is a random number wit an exponential distribution. (4) Channel loads are well balanced, and th traffic intensity to every channel is assumed to be constant. These assumption allow the best performance for probabilistic routing to be evaluated.

The network may be modeled by a queueing network consisting of multi ple M/M/1 queues. Let n_i be the number of packets queued at the i-th chann (including the one in transmission), and p_y be the probability that n_i is equal t y. The probability that there are k wasted channels may be formulated a follows:

$$P_r(W=k) = \frac{t!}{k!(t-k)!}\, p_0^k\, P_r(n_1+n_2+....+n_{t-k} \geq t \mid n_i \geq 1, 1 \leq i \leq t-k) \qquad (5.4)$$

$$+ \sum_{j=k+1}^{t-1} \frac{t!}{j!(t-j)!}\, p_0^j\, P_r(n_1+n_2+....+n_{t-j}=k+t-j \mid n_i \geq 1, 1 \leq i \leq t-j)$$

where p_0 is the probability of a channel being idle. The first term in the RHS of Eq. (5.4) represents the probability that the number of waiting requests is greater than or equal to the number of idle channels. According to the definition, all the idle channels are wasted. The second term, on the other hand, represents the probability that the number of waiting requests is less than the number of idle channels.

The set of events that the sum of a set of numbers being greater than t is mutually exclusive to the set of events that such a sum is never greater than t. Therefore, the following relation holds.

$$P_r(n_1+n_2+ \cdots +n_{t-k} \geq t \mid n_i \geq 1, 1 \leq i \leq t-k) \qquad (5.5)$$

$$= 1 - \sum_{s=1}^{t-1} P_r(n_1+n_2+ \cdots +n_{t-k}=s \mid n_i \geq 1, 1 \leq i \leq t-k)$$

The set of events that the total number of waiting requests is equal to r, $r \leq t$ includes all the combinations that the sum of waiting requests is r, i.e.,

$$P_r(n_1+n_2+ \cdots +n_{t-k}=s \mid n_j \geq 1, 1 \leq j \leq t-k)$$

$$= \sum_{(\text{combinations } n_1+n_2+ \cdots +n_{t-k}=s)} P_r(n_1 n_2 \cdots n_{t-k}) \qquad (5.6)$$

The left-hand-side of the equation, $n_1+n_2+ \cdots +n_{t-k}=s$, may be rearranged by grouping together those n_i's of equal value and represented in the following

form:

$$r = \sum_{i=1}^{s} m_i n_i \qquad (5.7$$

where m_i is the multiplicity of numbers, n_i, and s is the number of such group Accordingly, the arguments, s and m_i's, in this representation satisfys the fo lowing condition:

$$t-k = \sum_{i=1}^{s} m_i \qquad (5.7$$

With such a representation, the combinations that summation of u pos tive integers equals v, or $n_1+n_2+ \cdots +n_u=v$, may be obtained by applyin the algorithms proposed for generating u partitions of an integer v [REI79 Furthermore, since all queues operate independently, the following result ma be derived from queueing theory.

$$P_r(n_1,n_2, \cdots ,n_{t-k}) = p_{n_1} p_{n_2} \cdots p_{n_{t-k}} \qquad (5.8$$

For an M/M/1 queue, the probability of its load being n_i, denoted by p_{n_i}, has closed form solution that can be expressed in terms of its traffic intensity ρ.

$$p_{n_i} = \rho^{n_i}(1-\rho) \qquad (5.8$$

The mean value of W may be formulated by combining Eq's 5.4 to 5.8.

$$E(W) = \sum_{k=1}^{t-1} k * P_r(W=k)$$

$$= \sum_{k=1}^{t-1} k * \Bigg\{ \frac{t!}{k!(t-k)!}\, p_0^k\, \Big[(1-\rho)^k \rho^{t-k}$$

$$- \sum_{r=1}^{t-1} \sum_{\sum_{n=1}^{s} m_n q_n = r} \frac{(t-k)!}{m_1!m_2! \cdots m_s!} p_{q_1}^{m_1} p_{q_2}^{m_2} \cdots p_{q_s}^{m_s} \Bigg]$$

$$+ \sum_{j=k+1}^{t-1} \frac{t!}{j!(t-j)!} p_0^j$$

$$* \left[\sum_{(m_1 q_1 + m_2 q_2 + \cdots + m_v q_v = k+t-j)} \frac{(k+t-j)!}{m_1!m_2! \cdots m_s!} p_{q_1}^{m_1} p_{q_2}^{m_2} \cdots p_{q_v}^{m_v} \right] \Bigg\} \qquad (5.9)$$

Note that s and v in Eq. 5.9 are determined by Eq. 5.7. The percentage of the channels being wasted, which is defined as $\frac{E(W)}{t} \times 100\%$, is plotted against the traffic intensity as depicted in Figure 5.4. It is observed from this figure that probabilistic routing does not perform well when its load is moderate. Under this condition, there are more than 50% of the channels being wasted on the average. When ρ is small, there are only a few requests being blocked. Likewise, when ρ is large, there are not many idle channels. In both cases, the average number of channels being wasted is small as expected.

5.3.3 State-Dependent Routing

Probabilistic routing controls the branching probabilities according to the average behavior of the queueing system. Thus, it is unable to adapt to the instantaneous variations in load, and its performance is poor when the variance of the loads is large. A state dependent routing strategy, on the other hand, copes with the problem by determining the routing on a per-task basis depending on the load of every channel. With a state dependent routing, a request is

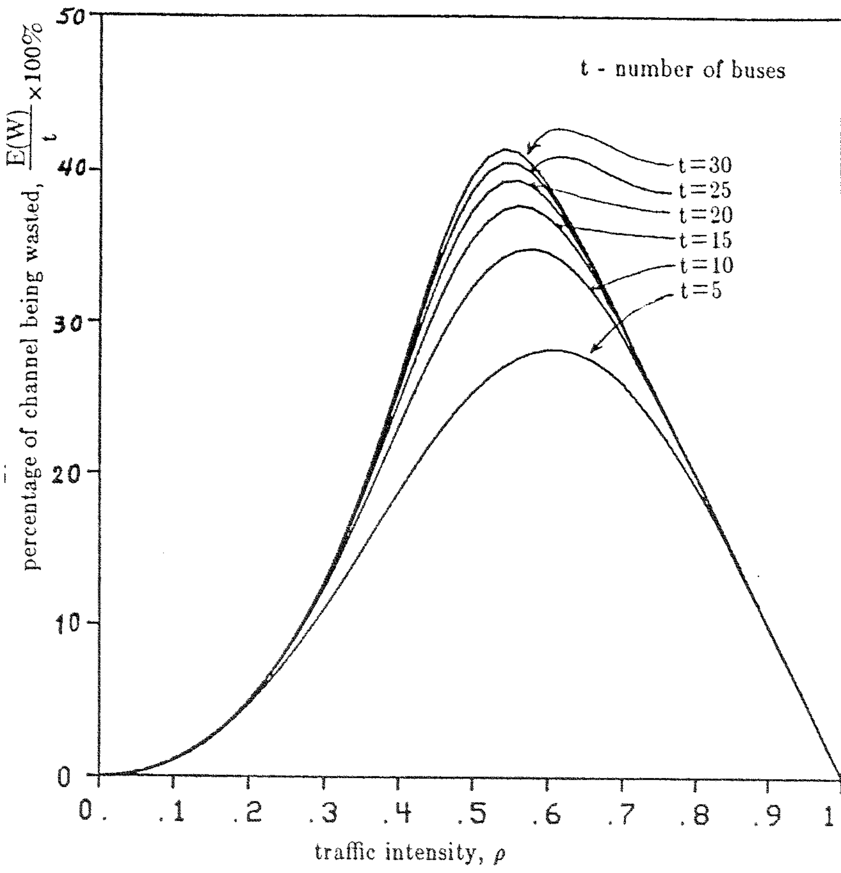


Figure 5.4 The percentage of channel being wasted (probabilistic routing)

routed to the channel with the minimum load. Let the number of processors contending for channel i be denoted by n_i. Then, the decision of this routing strategy is to identify the minimum among the n_i's, $1 \leq i \leq t$.

No good analytical technique has been developed for investigating the performance of state dependent routing strategies. A simulation approach is used here with the results plotted in Figure 5.5 using solid lines. It shows that the percentage of channels being wasted is reduced to less than 5%. In contrast to probabilistic routing, the percentage of channels wasted in state dependent routing decreases as the total number of channels in the network increases. This results in a larger degree of resource sharing. Note that more than one request may be generated before the channel load information is updated. These requests are routed to the same channel if they arrived at different processors. According to the simulation results, the effect of such an improper routing is minor unless there are bulk arrivals.

In a distributed computer system, the instantaneous load of a channel is not available to processors. Collecting this information by message transfers may introduce an unacceptable overhead. However, the channel loads may be estimated by methods as discussed in Chapters 3 and 4. Our simulation results are also plotted in Figure 5.5 using dashed lines. It shows that the bus utilization under this implementation is only slightly lower than the case in which all channel loads are known exactly. In conclusion, the state dependent routing with load estimation performs much better than an ideal probabilistic routing (Figure 5.6)

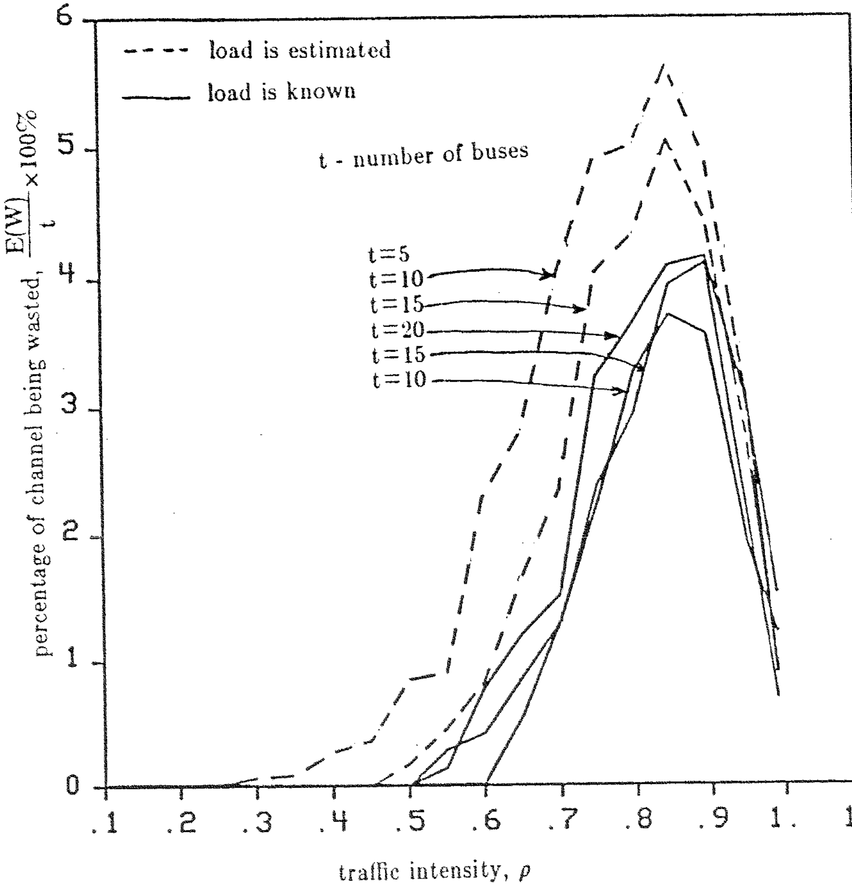


Figure 5.5 The percentage of channel being wasted (state-dependent routin

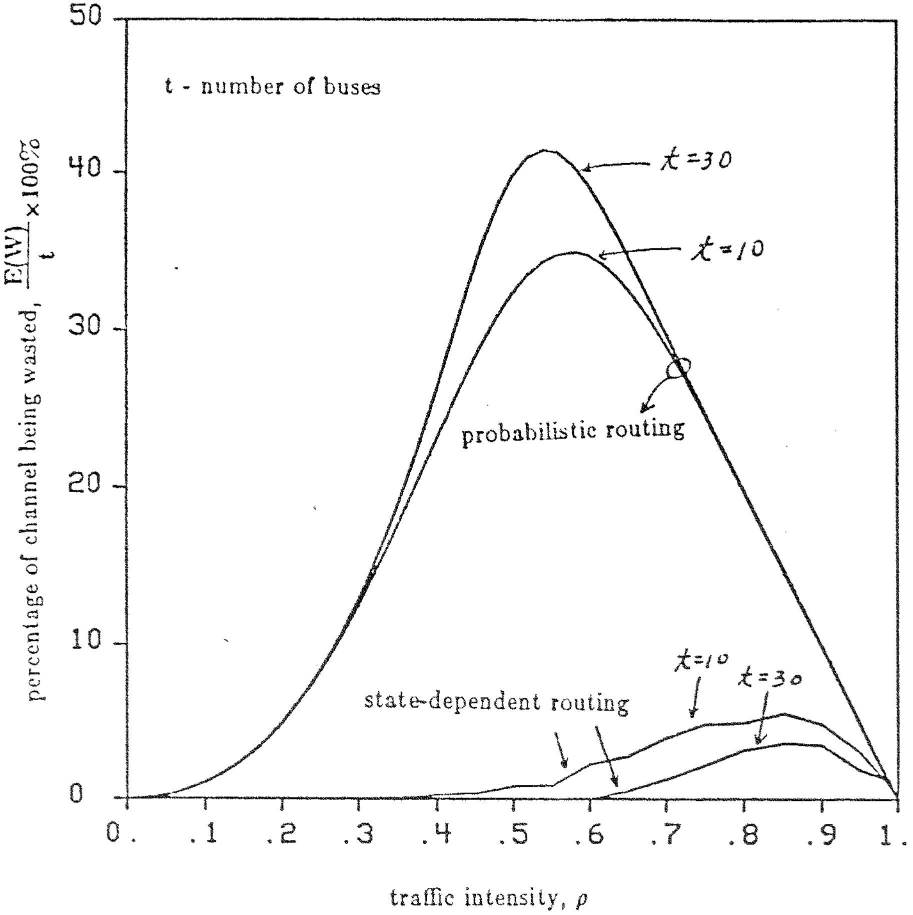


Figure 5.6 Comparison of probabilistic routing and state-dependent routing with load-estimation

5.4 Resource Allocation with Distributed Ordered Selection

Suppose N processors are bidding for resources through t channels. Eac processor is associated with a priority level representing the cost of being allc cated. Abstractly, the priority levels associated with these processors can b seen as a collection of N physically dispersed random number: $X_1 < X_2 < \cdots < X_t < \cdots < X_N$. A *(t,N)-selection* problem is to deter mine the t smallest numbers, $X_1, X_2, \cdots, X_t$, from the N random number: If the selected numbers have to be sorted, then the problem becomes th *ordered-selection* problem. Assuming that there exists an ordered-selection prc tocol, the scheduling of resource in a multiple-bus network can be carried ou as follows:

(1) Perform ordered selection to identify the t processors of the minimur cost.

(2) The i'th selected processor broadcasts its cost on the i'th channel.

(3) Perform ordered selection to identify the t resources of the minimur cost.

(4) The i'th selected resource broadcasts its cost on the i'th channel.

(5) Every selected processor computes a resource mapping locally using common algorithm. The mapping obtained should be identical since th same input data are used.

(6) Each selected processor send its request to the mapped resourc through a channel associated with the resource.

Two distributed ordered selection algorithms are developed and discussed below.

(i) Sequential Ordered Selection

An ordered selection can be done by iteratively finding the minimum from a set of distributed random numbers and excluding it from future iterations. It is obvious that the average time complexity of this approach is O(t) if the average time to find the minimum is constant. Constant time distributed minimum-search algorithm developed for single contention buses was discussed in Chapter 3. In adapting the algorithm for solving the ordered-selection problem, the central issue is to optimize the window control. However, the average time complexity remains to be O(t) as shown by Arrow [ARR81].

(ii) Parallel Ordered Selection

Alternatively, an ordered selection may be carried out in a multiple contention-bus network in parallel with a multi-window search scheme. The average time complexity will be shown to be O(logt) if t numbers are to be selected in a t-bus network. This scheme significantly improves over the sequential ordered selection schemes when t is large. The parallelism of the multi-window scheme lies on the capability of a station to listen to all buses simultaneously. It requires that each processor has a transceiver connected to each bus.

The basic idea of the multi-window search scheme is similar to that of the single-window search scheme. To search the minimum, sub-intervals of the search range are resolved by contention. Initially, the entire search range is

unresolved. An interval is determined for each bus as a transmission windc in each step of contention. As the contention proceeds, the search range is pa titioned into intervals. A given interval may be resolved or unresolved. resolved interval is one that may be either empty or contains exactly only o number. On the other hand, an unresolved interval is one that has never be searched or a collision was detected previously. The unresolved intervals a searched sequentially in a single window search scheme, while they are search in parallel in a multi-window search scheme. The multi-window search scher assigns to each bus an unresolved interval in a distributed fashion. A process transmits its request to a bus if its priority falls in the assigned interval that bus. A interval is resolved if there is no collision in contention. The pı cess is repeated until the intervals in the range that contains all the t small numbers are resolved. A multi-window ordered selection algorithm incurs additional message-transfer overhead since the information on the state o partitioned interval is available locally.

There are several issues regarding the implementation of such a schen First, the method of partitioning the search range into intervals in each step selection must be developed. Second, the time to terminate the selection p cedure has to be determined. Third, a way to sequence the selected numb must be provided. These issues are discussed in the following.

5.4.1 Multi-Window Control

The complexity of the multi-window control is much higher than that of the single-bus case. In the minimum-search algorithm developed for a single contention bus, the lower bound for the window can remain stationary. Optimization of window control is reduced to the determination of the window size (or upper bound) only. In the multi-window-search problem, both the bounds and the size of a search window can affect the efficiency of window control scheme.

(a) The Optimal Multi-Window Control

Multi-window control is to determine a set of windows for the next step of contentions. The windows are found from the set of unresolved intervals, including collided and unsearched ones. These intervals may be represented by a vector $\bar{V}$ in which each element represents an interval that consists of a triplet representing the starting point, the ending point, and the state of collision. Based on such a representation, the issue of the multi-window control optimization may be formulated in dynamic programming. The best choice of a window vector would be one that minimizes the number of subsequent contentions. This number depends on the results of subsequent contentions, and may be expressed by the average of all the possible outcomes.

Consider the case in which t numbers are to be selected from N distributed random numbers, and the current unresolved intervals are represented by a vector $\bar{V}$. Given $\bar{V}$, a contention step using contention windows $\bar{w}$ results in another unresolved status vector represented by $\bar{U}$. Denote the expected

number of contentions in performing the ordered selection by $C^t_{N,\bar{V}}$. Then th ordered selection process with an optimal window control may be expresse recursively as follows.

$$C^t_{N,\bar{V}} = \min_{\bar{w}} \left\{ 1 + \sum_{i=1}^{t}\sum_{\bar{U}} p_{N,t,i,\bar{V},\bar{U}}(\bar{w}) \times C^{t-i}_{N-i,\bar{U}} \right\} \qquad (5.12$$

where $p_{N,t,i,\bar{V},\bar{U}}(\bar{w})$ is the probability that the contention results in isolating numbers to be selected, and $\bar{U}$ is the status of unresolved intervals after con tention if window $\bar{w}$ is used. Contention for resolving the resulting unresolve intervals has to be continued until all the t minimum numbers are identified An optimal window vector is the one that minimize the recursive formulatio Eq. 5.12. In order to evaluate Eq. 5.12, all the $p_{N,t,i,\bar{V},\bar{U}}(\bar{w})$'s must be available which are too numerous to be evaluated. It is observed that there are 3^t possi ble outcomes of a contention. For each outcome, there is a corresponding vec tor $\bar{U}$, and for each vector $\bar{U}$, an optimal window $\bar{w}$ has to be determined. Ir each case, there are $\binom{t}{i}(t-i+d)^{(N-i)}$ possible distributions of X_i's in the inter vals, where d is the number of unsearched intervals. Hence the evaluation o Eq. 5.12 is too complex. We will pursue suboptimal solutions to the multi window control problem.

(b) A Heuristic Multi-Window Control

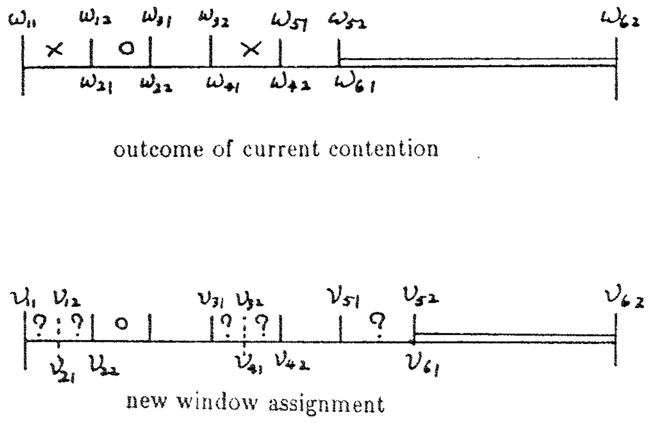
A k-window control has 2k degrees of freedom since there are two degree of freedom for each window: size and position. To simplify the multi-window control, the following restrictions are imposed in determining the windows. (1

The buses are allocated sequentially to search the unresolved intervals from the lower end to the upper end. The sequential allocation will leave a single un-searched interval at the upper-most end. (2) A collided interval is split into two equal halves, and each is considered as an unresolved interval. (3) Extend allocations into un-searched interval if all the collided intervals have been allocated. The window size of the window in the un-searched interval is $1/\hat{\gamma}$, where $\hat{\gamma}$ is the estimated number of X_i's falling in the unsearched interval. The size of unsearched range and the status of resolved and unresolved intervals may be used in estimating $\hat{\gamma}$. A simple estimation can be obtained by subtracting the estimated number of X_i's in the searched range from the total number of X_i's, i.e., setting $\hat{\gamma}$ to $N - q_t - 2 * \sum_{j=1}^{t} d_j$, where q_t and d_j will be defined in Section 5.4.2.

An example illustrating the above window control is shown in Figure 5.7. This figure shows that previous contentions result in isolating one number, excluding several empty intervals from further considerations, and revealing that there are two collided windows. According to the bus-assignment constraints, the search windows to be allocated to five buses for the next contention is shown in Figure 5.7b.

5.4.2 Sequencing The Selected Numbers

If the status of every interval is known, then the order of the selected numbers can be determined. However, the number of intervals grows as the contention proceeds. It may be too expensive for each processor to keep track of the status of all the intervals, and is not effective if the task is assigned to a



outcome of current contention

new window assignment

"×" - a collided interval
'O' - a success window
" " - an empty window
"=" - an unsearched interval
"?" - to be determined by contention

Figure 5.7 Illustration of heuristic window control

centralized station. To reduce the complexity of local processing, each processor can observe the contention activities continuously and counts the cumulative number of resolved intervals with upper bounds smaller than the local parameter. To achieve this, each processor has to maintain a vector consisting of the relative order of each contention window. It is updated according to the collision information after each contention. Let the collision information of a contention be represented by two t-tuples, $(s_1, s_2, \cdots, s_t)$ and $(d_1, d_2, \cdots, d_t)$, where

$$s_i = \begin{cases} 1 & \text{if contention in the i' th bus succeeds,} \\ 0 & \text{otherwise.} \end{cases}$$

$$d_i = \begin{cases} 1 & \text{if contention in the i' th bus collides,} \\ 0 & \text{otherwise.} \end{cases}$$

The order of the random numbers in contention windows are maintained in each processor by a vector

$\overline{Q} = (q_1, q_2, \cdots, q_t)$. These q_i's are set to 1 initially, and updated after each step of contention in the following way.

$$q_i \leftarrow q_i + \sum_{j=1}^{i-1} s_j \qquad \text{for } i=1,2,\cdots,t \tag{5.13}$$

It is easy to verify that the counts q_i's are the order in the ordered statistics when the selection procedure terminates.

5.4.3 The Termination Condition

To determine the termination condition, the status of all the intervals ma

have to be maintained. As an alternative, each processor has to maintain a

indicator that points to the position where the ordered statistics X_t and X_{t+}

are most likely to be separated. The indicator for termination can be set t

the upper end of the k-th window such that the following conditions hold.

$$q_k + 2 * \sum_{i=1}^{k} d_i \geq t > q_{k-1} + 2 * \sum_{i=1}^{k-1} d_i \qquad (5.1$$

From Eq. 5.13 we know that the order of random numbers in the i'th windov

q_i, is the cumulative number of contending parameters that have been isolate

and are smaller than the numbers in this window since the beginning of conte

tion. Thus, the selection process terminates correctly when every interv

smaller than the indicator is resolved, i.e. the termination condition may t

expressed as

$$\sum_{i=1}^{t} d_i = 0 \quad \text{and } q_t \geq t \qquad (5.1$$

Since the termination indicator is set in a way that the t minimu

numbers are always smaller than it, the termination indicator is also the upp

bound of the search range. Hence, the search range can be narrowed down

the contentions proceed.

5.5 A Multi-Window Search Scheme for Ordered Selection

A multiple-bus-CSMA/CD protocol based on the proposed multi-window search scheme with a heuristic window control is outlined in Figures 5.8 and 5.9. The statements in this protocol are implementation independent except for two functions : transmit(X, $\vec{W}$) and observe($\vec{S}$, $\vec{D}$). These two functions depend on the architecture of the contention buses. The procedure can be implemented on any type of networks that support these two functions. The logic of this procedure is also simple enough to allow it be implemented in hardware.

5.5.1 Proof of Correctness

We prove that the above multi-window search protocol correctly selects the t smallest numbers from N distributed random numbers. First, we show that the distributed procedure terminates in a finite number of steps.

Lemma 5.1 : The multi-window search protocol terminates in a finite number of steps if any pair of random numbers is separable. Two random numbers, x_i and x_j, are separable if $\frac{1}{N \times \left| x_i - x_j \right|}$ is finite.

<Proof>

The procedure terminates when all disjoint intervals within the search range are resolved. It remains to show that number of such intervals is finite, and these intervals were obtained by partitioning the search range in a finite number of steps.

```
procedure multi-window-search(X, X-order);
/* N : Total number of distributed random numbers */
/* t : number of random numbers to be selected */
/* L & U : Lower & upper bounds of the range of random numbers */
/* X : The random number generated by the local processor */
/* X-order : The order of X among the N distributed random numbers */
/* $\vec{W} = (\vec{w}_1, \vec{w}_2, \cdots, \vec{w}_t)$ : vector of search windows, where $w_r = (w_{1,1}, w_{1,2})$ */
/* $\vec{S}$ & $\vec{D}$ : Collision detection vectors */
/* $\vec{Q}$ : The estimated order of $Y_i$'s in the windows */
/* T : Termination indicator */
begin
        for i=1 to t do
        begin
                $s_i \leftarrow 0$; $d_i \leftarrow 0$; $q_i \leftarrow 1$;
        end;
        T ← U; X-order ← 1; done ← false;
        while ((not done) and (X-order $\leq$ t)) do
        begin
                window($\vec{W}$ , $\vec{Q}$ , $\vec{S}$ , $\vec{D}$ , T); /* Determine the transmission windows */
                transmit(X , $\vec{W}$); /* Transmit to k-th channel if $w_{k,1} < X \leq w_{k,2}$. */
                observe($\vec{S}$ , $\vec{D}$); /* Detect the outcome of the contention. */
                /* Update current order of each window and X-order. */
                for k=1 to (t+1) do
                begin
                        $q_k \leftarrow q_k + \sum_{j=1}^{k-1} s_j$;
                        if ( $X \geq w_{k,2}$ ) then X-order ← $q_k + s_k$;
                end;
                /* update search range by resetting termination indicator */
                i ← 0; r ← 0;
                while ((r < t) and (i < t)) do
                begin
                        i ← i+1; r ← $q_i + 2 * \sum_{j=1}^{i} d_j$;
                end;
                if ( i < t ) then
                begin
                        T ← $w_{i,2}$;
                        /* Termination ? */
                        if ( $\sum_{j=1}^{i} d_j = 0$) then done ← true;
                end;
        end;
end multi-window-search.
```

Figure 5.8 A multi-window ordered selection procedure

```
procedure window ($\vec{W},\vec{Q},\vec{S},\vec{D}$,T);
/* N : Total number of distributed random numbers */
/* t : Number. of random numbers to be selected */
/* L & U : Lower & upper bounds on the range of random numbers supplied from calling
function*/
/* $\vec{W} = (\vec{w}_1,\vec{w}_2, \cdots ,\vec{w}_t)$ : vector of search windows, where $w_f = (w_{i,1},w_{i,2})$ */
/* new-$\vec{W}$ : Temporary vector of new search windows */
/* $\vec{S}$ & $\vec{D}$ : Collision detection vectors */
/* $\vec{Q}$ : The estimated order of $y_i$'s in the windows */
/* T : Termination marker */

begin
        i ← 1;
        j ← 1;
        while ( i≤t ) do
                begin
                while ( $d_j$=0 ) do j ←j+1;
                if ( T≤U ) then
                        begin
                        /* Allocate two buses to resolve a collided interval */
                        new-$w_{i,1}$ ← $w_{j,1}$;
                        new-$w_{i,2}$ ← $\frac{w_{j,1}+w_{j,2}}{2}$;
                        new-$q_i$ ← $q_j$;
                        i ← i+1;
                        new-$w_{i,1}$ ← new−$w_{i-1,2}$;
                        new-$w_{i,2}$ ← $w_{j,2}$;
                        new-$q_i$ ← $q_j$;
                        end;
                else begin
                /* Search into unsearched interval */
                        for k=i to t do
                                begin
                                new-$w_{i,1}$ ← new−$w_{i-1,2}$;
                                new-$w_{i,2}$ ← $w_{t+1,1}+\frac{w_{t+1,2}-w_{t+1,1}}{N-q_t-2*\sum_{j=1}^{t}d_j}$;
                                end;
                        i ← t;
                        end;
        end;
        $\vec{W}$ ← new−$\vec{W}$;
        $\vec{Q}$ ← new−$\vec{Q}$;
end.
```

Figure 5.9 A heuristic window-control procedure

To search for the minimum in an unsearched range, the window control determines intervals of non-zero size as contention windows. The search range is only partitioned into a finite number of intervals. These intervals may be resolved or remain unresolved after a contention. If an interva remains unresolved, it is split into two halves of finite sizes. In the worst case, the number of steps to separate any two random numbers is

$$k = \log_2 \left\lceil \frac{1}{N * \delta} \right\rceil$$

where $\delta = \min \{ |x_i - x_j|, 1 \leq i,j, \leq t+1, \text{ and } i \neq j \}$. Since δ is finite, the valu of k is also finite and the interval can be resolved in finite steps. Thus the procedure terminates in a finite number of steps. □

If two numbers to be selected are unseparable, they will always fall in th same window. The contention outcome of this window will always be a colli sion, and the search procedure will keep splitting the window but unable t resolve the contention. Therefore, the procedure runs into an infinite loop Theoretically, the probability of two continuous random numbers bein unseparable is zero. In practice, two numbers may be identical because of limited number of bits used in binary representations, or identical priority level are associated with the processors that generated these random numbers. T prevent the procedure from running into an infinite loop, a small randon number may be added to each contending number in each contention t separate unseparable numbers. The random number should be chosen from small range so that the modification to the original contending numbers do not affect the physical meaning of the underlying applications. Stations ma use distinct sequences of random numbers for the modifications to eliminate the possibility of unseparability.

The following lemma is also necessary for the proof of correctness of the protocol.

Lemma 5.2 : All the intervals within the search range are resolved when the procedure terminates.

<Proof>

There are two cases in allocating contention buses to resolve unresolved intervals after a contention. In the first case, the following inequality holds.

$$(q_t-1)+2 * \sum_{j=1}^{t} d_j \ < t \tag{5.17}$$

Since $q_t \geq 1$, we have $2 * \sum_{j=1}^{t} d_j \ < t$. The number of unresolved intervals in the search range is less than the number of buses available. Thus, there are a sufficient number of buses to be allocated to all collided intervals, and the remaining buses will be used ,to extend the search into the unsearched range if it exists.

$$(q_t-1)+2 * \sum_{j=1}^{t} d_j \ \geq t \tag{5.18}$$

As the contention proceeds, and the second case will happen eventually. This is a case in which there are more intervals to be resolved than the number of buses available. In this case, there exists an index k, $1 \leq k \leq t$,

such that

$$(q_k-1)+2 * \sum_{j=1}^{k} d_j = \begin{cases} t & \text{if contention in the k'th bus succeeds} \\ t+1 & \text{if contention in the k'th bus causes collisi} \end{cases}$$

According to the termination control, the termination indicator is set w_k in this case. The total number of buses required to search all collide intervals within the range is less than t if $q_k > 1$ or $2 * \sum_{j=i}^{k} d_j = t$, ar there will be an unallocated interval if $2 * \sum_{j=i}^{k} d_j = t+1$. However, it w be excluded from the current contention if the outcome of collision dete tion does not satisfy the following conditions.

$$2 * \sum_{i=1}^{t-1} d_i = t-1 \ \text{ and } \ s_t d_t = 0$$

Or, if the condition is satisfied, the t'th bus will be allocated to search tl interval in the next contention. Hence, every interval within the sear range is searched until all intervals are resolved. □

Now, the proof of correctness of the proposed multi-window search procedure summarized in the following theorem.

Theorem 5.2: The multi-window search procedure with the proposed subo timal window control performs a (t,N)-selection correctly.

<Proof>

We have shown that the procedure locates the ordered statistics $X_1,X_2, \cdots$, and X_t correctly in a finite number of steps in Lemma 5.1. According to Lemma 5.2, all intervals below the termination indicator are resolved. From the way that the termination indicator is set, it is easy to show that there are at least t numbers being isolated in these intervals. Since the resolved intervals are disjoint and follow the linear ordering relation, the numbers isolated in these intervals can be ordered accordingly.

□

5.5.2 Performance of The Protocol

Splitting a collided interval into two equal halves is a good heuristic window control and is justified in this section. In Chapter 3, we have shown that the binary-divide window control is optimal for resolving a collided interval if there are exactly two numbers uniformly distributed in the interval. Suppose that the size of unsearched interval is u and that the estimated amount of random numbers falling in this intervals is $\hat{\gamma}$. Then the proposed multi-window control will allocate a bus to search a interval of size $\frac{u}{\hat{\gamma}}$. Let Z be the random variable representing the number of X_i's in the collided window, and the average number of X_i's falling in this search interval be E(Z).

$$E(Z) = \sum_{p=2}^{\hat{\gamma}} p * \binom{\hat{\gamma}}{p} \left(\frac{1}{\hat{\gamma}}\right)^p \left(1-\frac{1}{\hat{\gamma}}\right)^{\hat{\gamma}-p} \qquad (5.19)$$

The values of E(Z) with respect to different $\hat{\gamma}$'s are given in Table 5.1. The table shows that E(Z) is less than 2.4 for reasonably large values of $\hat{\gamma}$. A

Table 5.1 E(Z), the expected number of X_i's falling in a window of size $\frac{u}{\hat{\gamma}}$ when the estimated number of random numbers uniformly distributed in an unsearched interval (of size u) is $\hat{\gamma}$

$\hat{\gamma}$	E(Z)
2	2.0
5	2.312
10	2.326
20	2.329
100	2.337
500	2.346
1000	2.379

collided window contains less than 2.4 random numbers on the average since the number of X_i's in a collided window that has been split would be smaller than this value. The arguments above concludes that a collided window often contains two numbers. Hence, the binary-divide rule is a good heuristic to separate these two numbers.

The following lemma and theorem show that the third constraint of the heuristic window control also results an efficient window control.

Lemma 5.3 : Let p be a natural number, and $y_i > 0$. If $\sum_{i=1}^{p} y_i = c$, then $\prod_{i=1}^{p} y_i$ is maximized when $y_i = \frac{c}{p}$, i=1,2,...p.

<Proof>

The lemma is proved by a mathematical induction. The induction base (p=1) is trivial. Consider the cases in which p is greater than one. Assume that

$$\sum_{i=1}^{p-1} y_i = \beta c \qquad \text{where } 0<\beta<1 \tag{5.20}$$

Accordingly, $y_p = (1\text{-}\beta)c$. If p is equal to 2, then

$$y_1 y_2 = c^2\beta(1-\beta) \tag{5.21}$$

The right-hand side of Eq. 5,21 is maximized when $\beta = \frac{1}{2}$. So the lemma is also true for p=2.

Assume that the lemma is true for p = m so that $y_i = \frac{c}{m}$. Consider the case in which p = m+1. Let $y_{m+1} = (1-\beta)c$, then $\sum_{i=1}^{m} y_i = \beta c$ is maximized

at $y_i = \beta\frac{c}{m}$. Hence

$$\prod_{i=1}^{m+1} y_i = (\prod_{i=1}^{m} y_i) \times y_{m+1} = \left(\beta\frac{c}{m}\right)^m \left(1\text{-}\beta\right)c \qquad (5.22$$

The RHS of Eq. 5.22 is maximized at $\beta = \frac{m}{m+1}$. This yield

$y_i = \frac{c}{m+1}$, for i=1,2,...,m+1. □

Theorem 5.3: Let the number of X_i's distributed in the unsearched range (0,
be n, and the boundaries of windows be (w_{i-1}, w_i), i=1,2,...t. Then, the prob
bility that all t of X_i's are isolated within one contention is maximized whe
$w_i - w_{i-1} = \frac{r}{n}$, i=2,..., t.

<Proof>

Without loss of generality, we assume that all random number are ger
erated from a uniform distribution in the interval (0,1). The joint proba
bility density function of the ordered statistics, $y_1, y_2, \ldots, y_{t+1}$, in general, is

$$f_{y_1 y_2, \ldots, y_{t+1}}(x_1, x_2, \ . \ . \ . \ , x_{t+1}) =$$

$$\frac{n!}{(n-t-1)!} g(x_1) g(x_2) \cdots g(x_{t+1})(1-G(x_{t+1}))^{n-t-1} \qquad (5.2$$

where g(x) and G(x) are probability density function and distributic
function of the parent distribution respectively. But for uniform distrib
tion, we have g(x) = 1 and G(x) = x. So,

$$f_{y_1 y_2 \ldots y_{t+1}}(x_1, x_2, \ldots, x_{t+1}) = \frac{n!}{(n-t-1)!}(1-x_{t+1})^{n-t-1} \qquad (5.24)$$

A selection is successful when $y_i \in (w_{i-1}, w_i)$, $i=1,2,\ldots,t$, and $y_{t+1} \in (w_t, 1)$. Then the probability of success may be expressed in the following equation.

$$\Pr\{y_i \in (w_{i-1}, w_i), i=1,2,\ldots t, \text{ and } y_{t+1} \in (w_t, 1)\}$$

$$= \int_0^{w_1}\int_{w_1}^{w_2} \cdots \int_{w_{t-1}}^{w_t}\int_{w_t}^{1} \frac{n!}{(n-t-1)!}(1-x_{t+1})^{n-t-1} dx_{t+1} dx_t \ldots dx_2 dx_1$$

$$= \frac{n!}{(n-t)!}(1-w_t)^{n-t}\prod_{i=1}^{t}(w_i - w_{i-1}) \qquad (5.25)$$

Notice that the value of $\prod_{i=1}^{t}(w_i - w_{i-1})$ depends on w_t and the partitioning of the interval $(0, w_t)$. It follows from Lemma 5.3 that $\prod_{i=1}^{t}(w_i - w_{i-1})$ is maximized when $w_i - w_{i-1} = \frac{w_t}{t}$. Substituting it into Eq. 5.25 yields

$$\Pr\{y_i \in (w_{i-1}, w_i), i=1,2,\ldots t, \text{ and } y_{t+1} \in (w_t, 1)\}$$

$$= \frac{n!}{(n-t)!}(1-w_t)^{n-t}\left(\frac{w_t}{t}\right)^t \qquad (5.26)$$

It is easy to very that Eq. 5.26 is maximized at $w_t = \frac{t}{n}$. This leads to the results of $w_i = \frac{i}{n}$.

The w_i's have to be adjusted by a factor r if the random numbers are distributed in the interval $(0,r)$, Thus, $w_i - w_{i-1} = \frac{r}{n}$. □

A simulation study has been conducted to evaluate the performance of th multi-window ordered selection utilizing the proposed heuristic window-contr scheme. The simulation results are plotted in Figure 5.10. It shows that th average number of contentions in resolving (t,N) ordered selections is propor tional to $\log_e t$. To reveal this fact, dashed lines showing the functio $\log_e t + 3.5$ are also plotted in this figure. It is shown in Figure 5.11 that th performance of the multi-window selection is independent of N.

5.6 Concluding Remarks

In this chapter, we have shown that optimal resource scheduling problem ma be reduced to the ordered selection problem if the scheduling discipline satisfie the independence relation. A multi-window search scheme is proposed to per form ordered selection on multiple contention-bus. The proposed heuristi determines the smallest t numbers from a set of distributed random numbers i O(log t) contention slots.

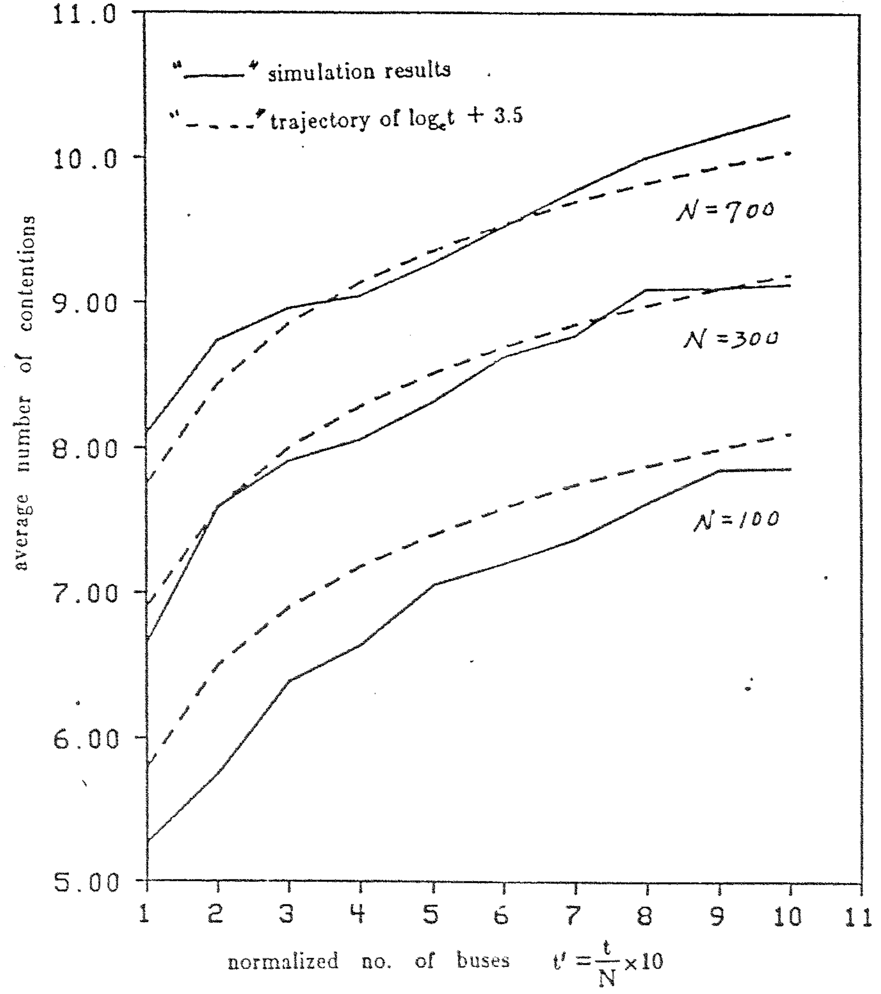


Figure 5.10 The performance of multi-window ordered selection

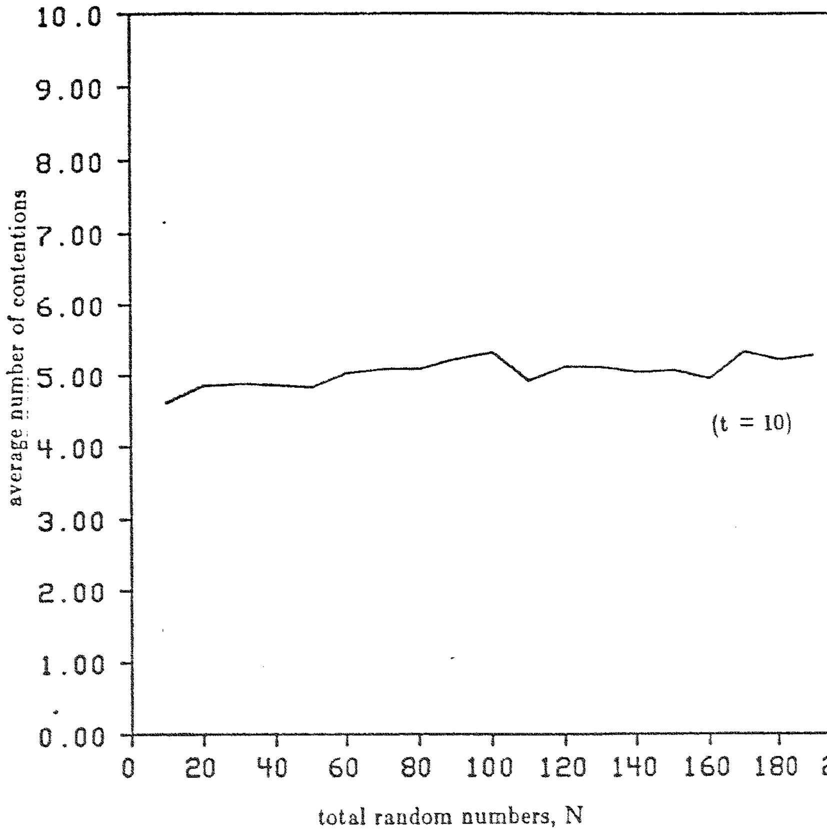


Figure 5.11 The performance of multi-window ordered selection with fix
number of buses

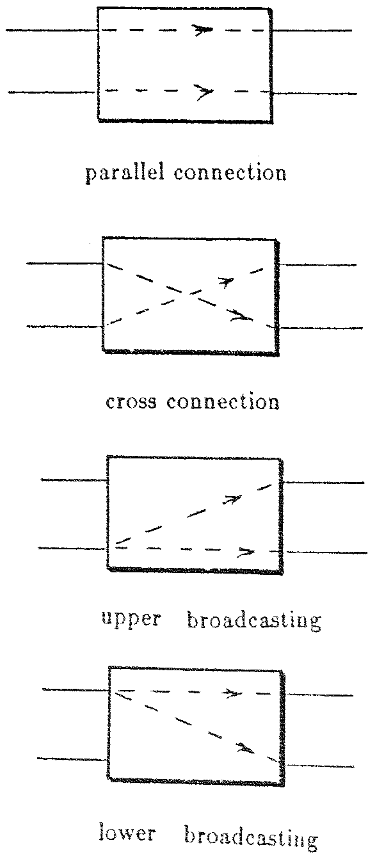
Table 5.2 The design of the distributed resource allocation scheme in multiple contention-bus networks

Single Bus	t Buses
One Request of the Highest Priority	t Requests of the Highest Priority
Most Preferable Resources	t Most preferable Resources
Distributed Minimum Search	Distributed Ordered Selection
Window Search	Multi-Window Search
Collision Detection	Collision Detection
No Explicit Message Transfer	No Explicit Message Transfer
2.4 Contention Slots	O($\log_2 t$) Contention Slots

CHAPTER VI
RESOURCE ALLOCATION IN MULTISTAGE INTERCONNECTION NETWORKS

A multistage interconnection network is composed of a set of a×b cross
bar switches (usually 2×2) that are organized into stages as illustrated by a
Omega network in Figure 6.2 [LAW75]. A switch may be in one of the fou
connection states as shown in Figure 6.1. By setting the switches properly
each stage may realize a permutation from its inputs to its outputs. A map
ping from processors to resources may be obtained through the composition c
multiple permutations. Both centralized and distributed switching control hav
been developed for multistage interconnection networks. It has been show
that this type of interconnection networks are most cost-effective for intercon
necting processors and memories in SIMD array processors or MIMD multipro
cessors [PAT81, STO71, SIE79a, SIE79b, KRU83]. They provide ampl
bandwidth while using much less hardware than crossbar switches.

It is the objective of this chapter to study the resource allocation proble
on this type of networks. The complex interconnection structure complicate
the problem of representing network constraints in optimization. We will sho
that the optimal resource mapping may be obtained by transforming the prob
lem into a corresponding network-flow problem and solving it by existing alg
rithms.



parallel connection

cross connection

upper broadcasting

lower broadcasting

Figure 6.1 Different connection states of a 2×2 cross-bar switch

6.1 Multistage Resource Sharing Interconnection Networks

A *Multistage Resource Sharing Interconnection Network* is abbreviated MRSIN throughout this chapter. A MRSIN has a similar interconnecti structure but a different routing strategy as in conventional interconnecti networks with address mapping [FEN81]. In a conventional interconnecti network, the resource address has to be determined before a request enters t network [SIE79a, SIE81]. In a MRSIN, a request is directed to any f resource in the pool and does not require the destination address of t resource to be known before entering the network [WAH82, WAH8 WAH84b, JUA84c].

The basic assumptions made in this chapter are:

(a) Circuit switching is assumed for the MRSINs rather than pac switching for the following reasons. First, packet switching is used in conv tional address mapping networks because it allows a network path to be shar by more than one request concurrently. This reduces the waiting time accessing a resource. The issue is less critical in MRSINs because a request c always search for another available resource provided that the network is fr Further, the overhead of rerouting a packet when a path or resource is block is higher than that of rerouting a resource request. Second, due to the char teristics of resources, a task cannot be processed until it is completely receiv The extra delay in breaking a task into multiple packets may decrease the ut ization of resources, and hence increase the response time of the system.

(b) One or more types of resources may exist. A MRSIN is classified the number of types of resources connected. A MRSIN with only one type resources attached is *homogeneous*, while a MRSIN with multiple types of resources is *heterogeneous*.

(c) Request priority may be associated with a request to indicate the urgency of the request. Resource preference may be associated with a resource to indicate the desirability of being used for service.

(d) Each request needs one resource only.

6.2 Resource Scheduling in Multistage Interconnection Networks

Blockage in conventional interconnection networks may be due to blockage in the destination or blockage in the network links. In a MRSIN, blockage in a resource can be avoided by searching for an alternate free resource. However, this may not always lead to better resource utilization because the allocation of one request to a resource may block one or more other requests from accessing free resources due to the blocked paths. A scheduling algorithm is, therefore, essential. For example, consider an 8-by-8 Omega network [LAW75] (Figure 6.2) with interchange boxes that can be set to one of the two possible states: straight or exchange (broadcast connection is not needed since one resource is needed for each request). In this example, assume that processors p_1, p_3, p_5 are requesting one resource each, and resources r_1, r_2, r_3 are available. Other processors are not making requests, and other resources are busy. Further, the network is completely free. All the resources will be allocated if the following processor-resource mappings are used: $\{(p_1, r_1), (p_3, r_2), (p_5, r_3)\}$, $\{(p_1, r_2), (p_3, r_1), (p_5, r_3)\}$, $\{(p_1, r_3), (p_3, r_1), (p_5, r_2)\}$ or $\{(p_1, r_3), (p_3, r_2), (p_5, r_1)\}$. But if

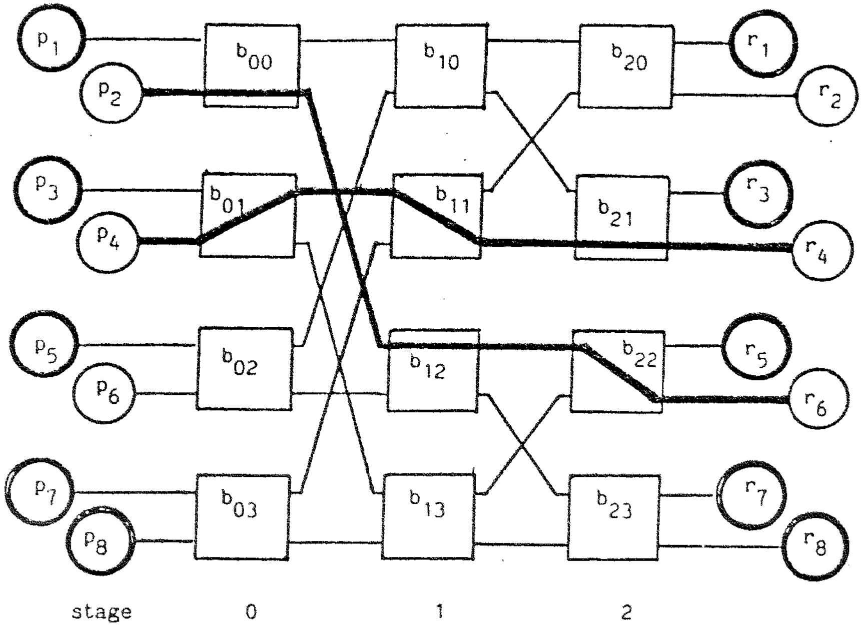


Figure 6.2 A multistage resource sharing interconnection network embedded in an 8×8 Omega network

the following processor-resource mappings are used: $\{(p_1, r_1), (p_3, r_3), (p_5, r_2)\}$ or $\{(p_1, r_2), (p_3, r_3), (p_5, r_1)\}$, then a maximum of two out of three resources can be allocated without blocking. This illustrates that the scheduler must be designed properly to give the maximum resource utilization. Simulations have shown that the average blocking probability can be as low as 2% for a MRSIN with an 8-by-8 cube network [HIC82,WAH82]. (A cube network is an Omega network with a permutation of the output ports [SIE81].) If a heuristic scheduling algorithm is used, then the average blocking probability increases to around 20%. Further degradation occurs if the network is not completely free.

Due to the complexity of the network configuration, efficient algorithm for solving this optimization problem cannot be done by simply identifying the processors of the maximum priority and resources of the maximum preference. In this chapter, we examine a subset of scheduling disciplines in which the cost of allocating a pair of processor and resource is the sum of the costs of the processor and the resource. Within this class of scheduling disciplines, the optimization problem may be solved by network-flow optimization algorithms.

6.3 Optimal Resource Scheduling for Homogeneous MRSINs

In this section, methods for optimizing resource mapping are discussed. Exhaustive methods that examines all the possible ordered mappings have exponential complexity. In a homogeneous MRSIN, suppose x processors are making requests, y resources are available, and the network is completely free. The scheduler has to try a maximum of $\begin{bmatrix}x\\y\end{bmatrix} y!$ (for $x \geq y$) or $\begin{bmatrix}y\\x\end{bmatrix} x!$ (for $y \geq x$)

mappings to find the best one [WAH82,HIC82]. Sub-optimal heuristics can l used, but is only practical when x and y are small. This problem is more cor plex when the network is partially occupied.

In this section, we transform the optimal resource mapping problem in various network-flow problems that can be solved by many efficient algorithn [GOL81]. The basic concepts of flow networks are briefly discussed here.

6.3.1. Flow Networks

A flow network is usually represented by a digraph with weighted link Let $\mathbf{D} = (\mathbf{V},\mathbf{E})$ be a digraph with distinct vertices s (*source*) and t (*sink*). capacity function, c(e), is defined on arc e of **D** such that c(e) be a non-negati real number for all $e \in \mathbf{E}$. A flow function f is an assignment of a real numb $f(e)$ to each arc, e, such that the following conditions hold:

(1) *Capacity limitation* : For every arc $e \in \mathbf{E}$,

$$0 \leq f(e) \leq c(e)$$

This condition ensures that a flow is less than the link capacity.

(2) *Flow conservation*: Let $\alpha(v)$ be the set of incoming arcs of vertex v, ar $\beta(v)$ be the set of outgoing arcs of vertex v. For every $v \in \mathbf{V}-\{s,t\}$,

$$\sum_{e \in \alpha(v)} f(e) = \sum_{e \in \beta(v)} f(e)$$

This constraint ensures that each intermediate node in the flow netwo does not absorb or create flows.

A *legal flow* is a flow assignment that satisfies capacity-limitation ar flow-conservation constraints. The ***network-flow problem*** is to find the leg flow that optimizes a given objective function. Examples include the maximum-flow, the minimum-cost flow, and the trans-shipment problems [PAP82]. An *s-t path* is a directed path from s to t. Insertion of a dummy node in a path will increase the path length, but will not affect the flow assignment. Increasing the length of s-t paths in this way so that all s-t paths are of equal length will be called an *s-t path equalization*. For convenience, all s-t paths are equalized in the following discussions.

The maximum-flow problem of a flow network, **G(V,E**,s,t,c), finds the maximum flow, F, from s to t under the capacity and flow-conservation constraints. It can be formulated as a linear programming problem:

Maximum-Flow Problem:

Maximize F

subject to:

$$(1)\quad \sum_{e\in\alpha(v)} f(e) - \sum_{e\in\beta(v)} f(e) = \begin{cases} -F & v = s \\ F & v = t \\ 0 & \text{otherwise} \end{cases} \qquad \text{(flow conservation)}$$

$$(2)\quad 0 \leq f(e) \leq c(e) \qquad \text{for all } e\in E \qquad \text{(capacity limitation)}$$

6.3.2. Optimal Resource Mapping in Homogeneous MRSINs

A switch-box in a MRSIN is an n-by-m crossbar switch, where $m,n \geq 1$. The non-broadcasting setting of the switch is equivalent to a legal integral flow assignment in a flow network of unit capacity. The following theorem assures that a valid resource mapping can be obtained by finding a legal integral flow in the corresponding flow network.

Theorem 6.1: For any MRSIN, there exists a flow network for which a lega integral flow is equivalent to a valid resource mapping.

<Proof>

Consider an n-by-m switch-box, where n is the number of input links an m is the number of output links. A physically realizable non-broadcastin setting of a switch is one in which an input link is connected to only on output link and vice versa. This switch-box is equivalent to a node in flow network with $|\alpha(v)| = n$ and $|\beta(v)| = m$. The switch setting equivalent to a legal integral flow assignment. Since a link in a switch either allocated or free while a flow assignment assigns a link to either zer or one unit of flow, the capacity constraints are always satisfied if th capacity of links is set to 1. There is a direct correspondence between switch-box and a node, and a switch setting and a flow assignment. So a equivalent flow can be constructed by a direct transformation from circuit-switched MRSIN. □

The following transformation produces a flow network such that th optimal resource mapping of a homogeneous MRSIN with random allocatio can be derived from its maximum-flow assignments.

Transformation 6.1: Create a flow network **G(V,E**,s,t,c) from a homogeneous MRSIN

T1. Create node-sets **P**, **X**, and **R** for processors, switch-boxes, and resourc respectively. Introduce two additional nodes: source s and sink t. Define

$$V' = \{s,t\} \cup P \cup X \cup R$$

T2. Add an arc from the source to each node of an associated processor. This set of arcs is denoted by **S**.

$$S = \{ (s,v) \mid v \in P \}$$

Add an arc from each node of an associated resource to the sink. This set of arcs is called **T**.

$$T = \{ (v,t) \mid v \in R \}$$

For each link in the MRSIN that connects two switch-boxes, a processor to a switch-box or a switch-box to a resource, add an arc between the corresponding nodes in the flow graph. Denote this set of arcs by **B**.

$$B = \{ (v,w) \mid v \in P \cup X, w \in X \cup R \}$$

Define:

$$E' = S \cup T \cup B$$

T3. Define capacity function c as follows:

$$\underset{e \in B}{c(e)} = \begin{cases} 0 & \text{associated link is occupied} \\ 1 & \text{associated link is free} \end{cases}$$

$$\underset{e \in S}{c(e)} = \begin{cases} 0 & \text{associated processor does not generate request} \\ 1 & \text{associated processor generates request} \end{cases}$$

$$\underset{e \in T}{c(e)} = \begin{cases} 0 & \text{associated resource is unavailable} \\ 1 & \text{associated resource is available} \end{cases}$$

T4. Obtain arc-set **E** by removing those arcs with zero capacity.

$$E = E' - \{ e \mid e \in E', c(e) = 0 \}$$

Obtain node-set **V** by deleting nodes that are not reachable from s.

$$V = V' - \{ v \mid \alpha(v) \cup \beta(v) = \emptyset, v \in V' \}$$

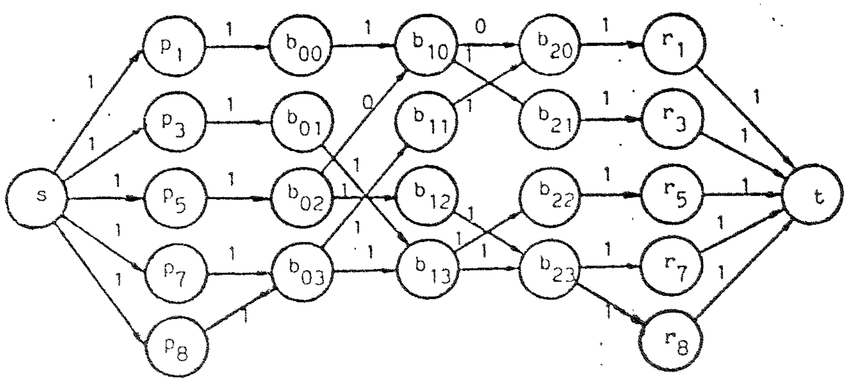
As an example, consider a MRSIN embedded in an 8-by-8 Omega netwo in which two paths are already occupied (Figure 6.2). Processors p_1, p_3, p_5, p and p_8 are making requests, and resources r_1, r_3, r_5, r_7, and r_8 are available. I applying Transformation 1, the resulting flow network is shown in Figure 6.3.

The correctness of the transformation is proved in the following theorem.

Theorem 6.2: In a homogeneous MRSIN with random allocation, the numb of resources allocated by a mapping is equal to the value of a legal integral flc in the corresponding flow network as obtained by Transformation 6.1.

<*Proof*>

Consider a flow network with equalized s-t paths. The integral flc assignment of nodes in Stage i is a mapping, g_i, that maps output flows Stage i−1 to input flows in Stage i+1. A legal integral flow assignment f the network can be represented by a composite function, $h = g_1 g_2 ... g$ where L is the length of the equalized s-t paths. For a flow network integral capacity as obtained by Transformation 6.1, it has been prov that a flow assigned by a maximum flow algorithm to an arc e, $f(e)$, either 0 or 1 [BON81]. Due to the conservation law, the incoming flow equal to the outgoing flow at node i. Hence, g_i is one-to-one. It follo that h is also a one-to-one function. For any one-to-one function, the nor of its domain is equal to the norm of its range. The norm of t



All arcs have capacity one

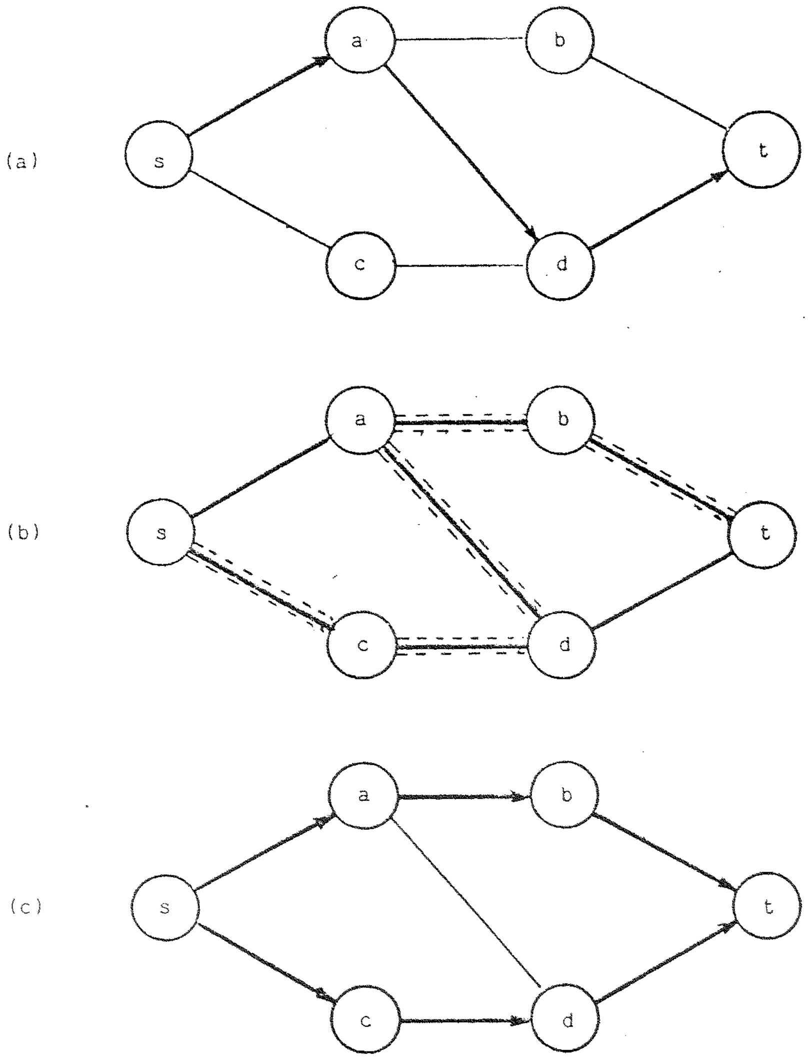
The number shown on each arc is the maximum flow assigned to the arc

Figure 6.3 The associated flow-network obtained from Figure 6.2 by Transformation 6.1

composite flow-assignment function, h, is equal to the total flow leaving the source and entering the sink. Thus, $|I| = |O| = F$, where I and O represent the domain and range of g_i, and F is the value of the flow. This implies that every legal flow defines a set of F nonoverlapping paths from s to t. In the MRSIN, each of these paths joins a pair of requesting processor and free resource such that the number of resources allocated is equal to the value of the corresponding flow in the flow network. □

From Theorem 6.2 and from a known result that the maximum flow of a network with integral capacity is integral [BON81], we conclude that ar integral maximum-flow of the transformed flow network is equivalent to the optimal resource mapping.

The flow assignment can be obtained by applying the maximum-flow algorithm. The first algorithm proposed for solving maximum flow problems is due to Ford and Fulkerson [FOR62]. It is a primal-dual algorithm in which the flow value is increased by iteratively looking for *flow augmenting paths* until a minimum cut-set is saturated. A flow augmenting path is a path from the source to the sink that can be used to increase the flow value and does no have to be a directed path. When an arc e on this path points in the directior from s to t, then additional flow may be pushed through e if the current flov assigned to e is less than its capacity, c(e). If arc e points in the opposite direction, then additional flow from s to t may be pushed through it if some of it flow is canceled. Thus, $f(e) \geq 0$ always holds. For example, in Figure 6.4a, ar original flow f is assigned along the path s-a-d-t. Then path s-c-d-a-b-t is flow augmenting path (Figure 6.4b). Advancing one unit of flow through thi augmenting path results in a new flow assignment f'. Two units of flow ar



(a) A Flow Network with an Initial Flow Assigned to Path s-a-d-t.

(b) An Augmenting Path s-c-d-a-b-t exists in this network.

(c) Final Flow after Advancing a Unit of Flow through the Augmenting Path

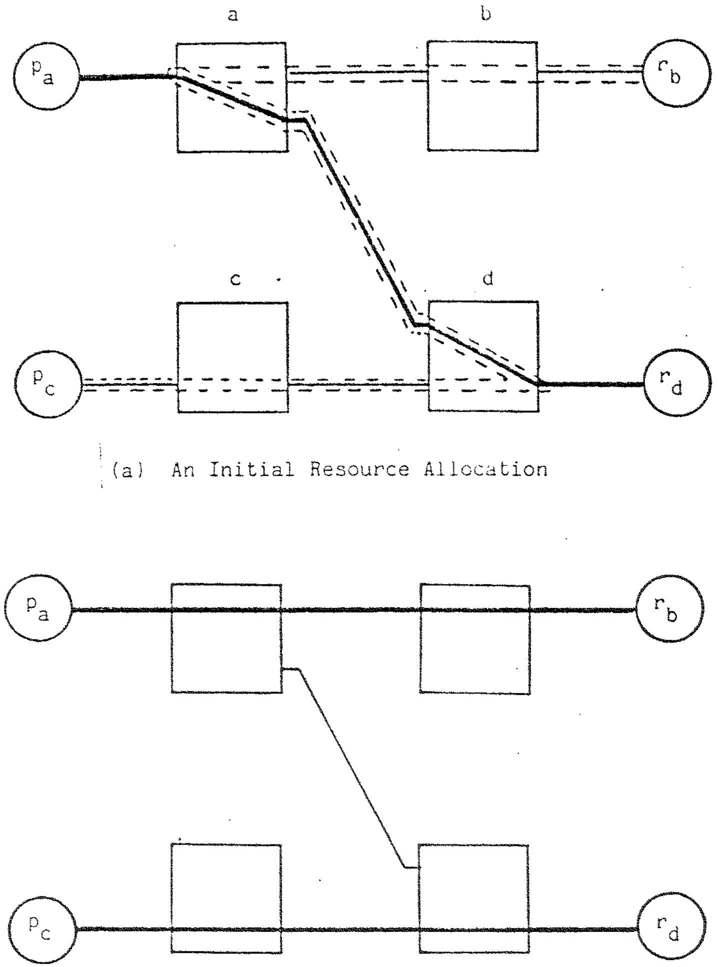
(All arcs have capacity 1)

Figure 6.4 Illustration of flow-augmenting paths

pushed through two separate paths s-a-b-t and s-c-d-t according to this assignment (Figure 6.4c).

In the MRSIN, advancing flow through an augmenting path is equivalent to a ***resource relocation***, i.e., a permutation of resource mapping. Consider th MRSIN in Figure 6.5a, which is the counterpart of the flow network used i the last example. The original flow f is equivalent to a resource mappin $\{(p_a,r_d), (p_c,y_b)\}$, while the existence of the flow augmenting path s-c-d-a-b- implies that there is a blockage due to this mapping. Advancing flow throug this path removes the blockage and results in a new mapping $\{(p_a,r_b), (p_c,r_d)$ as shown in Figure 6.5b. As an example, applying the network-flow algorith on the graph in Figure 6.3, the following processor-resource mappings ar obtained: $\{(p_1,r_3), (p_3,r_5), (p_5,r_8), (p_7,r_1), (p_8,r_7)\}$. There exists mappings tha cannot allocate all the free resources, but they are eliminated by the network flow algorithm. The flows in the graph are also shown in Figure 6.3.

Finding a flow augmenting path from the source to the sink in a flow network is the central idea in most maximum-flow algorithms. The improvemen lies in the efficient search of flow augmenting path [EDM72,DIN70]. For example, in Dinic's algorithm, the shortest augmenting path is always advanced firs with the aid of an auxiliary layered network, and the computational complexity is bounded by $O(|E|^3)$ for general networks. For a flow network of uni capacity as in our case, a even better time complexity is expected.



(a) An Initial Resource Allocation

(b) Two Resources are Allocated After Relocation.

Figure 6.5 Illustration of a resource relocation

6.3.3. Homogeneous MRSINs With Request Priority and Resource Preference

In a homogeneous MRSIN with request priority and resource preference, a request is associated with a priority level, and a resource is assigned a preference value. As discussed previously, many application-dependent attributes such as workload, execution speed, utilization, and capability can be encoded into the request priorities and resource preferences. The objectives of resource scheduling here is to maximize the number of resources allocated while allowing requests of higher priority to are allocated, and resources of higher preference to be chosen. However, it is not necessary that requests and resources are allocated in the order of priorities and preferences. Further, the allocation of a request of higher priority (cf. resource of higher preference) may block other requests of lower priority (cf. resources of lower preference). For this class of MRSINs, the resource mapping problem can be transformed into the *minimum-cost flow* problem.

Consider a flow network **G(V,E,s,t,c,w)** in which a cost per unit flow, w(e), is associated with edge e∈**E**. The *minimum-cost flow* problem finds a legal s-t flow assignment of value F_0 with the minimum cost. The constraints in the minimum-cost flow problem are the same as those for the maximum-flow problem. However, the objective function is to determine the cheapest s-t paths through which a fixed amount of flow F_0 can be pushed. The problem may be defined in a linear programming formulation:

Minimum-Cost Flow Problem

Minimize $\sum_{e\in E} w(e) f(e)$

subject to:

$$(1) \quad \sum_{e\in\alpha(v)} f(e) - \sum_{e\in\beta(v)} f(e) = \begin{cases} -F_0 & \text{if } v = s \\ F_0 & \text{if } v = t \\ 0 & \text{otherwise} \end{cases} \qquad \text{(flow conservation)}$$

$$(2) \quad 0 \leq f(e) \leq c(e) \qquad \text{for all } e\in E \qquad \text{(capacity limitation)}$$

In allocating resources, the objective is to find corresponding flow network whose optimal legal flows leads to an optimal resource mapping. The main idea behind the transformation is to embed priority and preference information into the objective function and the proper weight assignments on the links. A possible transformation is given as follows:

Transformation 6.2: Create a flow network **G(V,E**,s,t,c,w) from a homogeneous MRSIN with request priority and resource preference.

T1. Create node-sets **P**, **X**, and **R** for processors, switch-boxes, and resources, respectively, and introduce special nodes: source, s, sink, t, and a *bypass node*, u. Let:

$$V' = \{s,t,u\} \cup P \cup X \cup R$$

T2. Create arc-sets **S**, **T**, and **B** as in Step T2 of Transformation 6.1.

Add an arc from the node associated with a processor to the bypass node, and connect the bypass node to the sink. This set of arcs is denoted as **L**:

$$L = \{(v,u) \mid v\in P\} \cup \{(u,t)\}$$

Define:

$$E' = S \cup T \cup B \cup L$$

T3. Define capacity function c as in Step T3 of Transformation 6.1. In addition, define:

$$\underset{e\in L}{c(e)} = \begin{cases} 1 & e \neq (u,t) \\ \alpha(u) & e = (u,t) \end{cases}$$

T4. Define the cost function w that assigns the cost of carrying one unit of flow through a link e as follows:

$$w(e) = \begin{cases} 0 & \text{for } e\in B \\ \max(y_{max}+1, q_{max}+1) & \text{for } e\in L \\ y_{max} - y_p & \text{for } e\in S \\ q_{max} - q_s & \text{for } e\in T \end{cases}$$

where y_{max} is the highest priority level; y_p is the priority of the request from processor p; q_{max} is the highest preference level; and q_s is the preference of resource s.

T5. Create arc-set **E** and node-set **V** as in Step T4 of Transformation 6.1.

T6. Set the total flow F_0 to the number of requests.

As an example, in the MRSIN shown in Figure 6.6, each request is attributed with a priority level, and an available resource is given a preference value. The preference and priority levels range from 1 to 10. A minimum-cost flow network is obtained by applying Transformation 6.2 to the MRSIN and is shown in Figure 6.7.

The following theorem shows the correctness of Transformation 6.2.

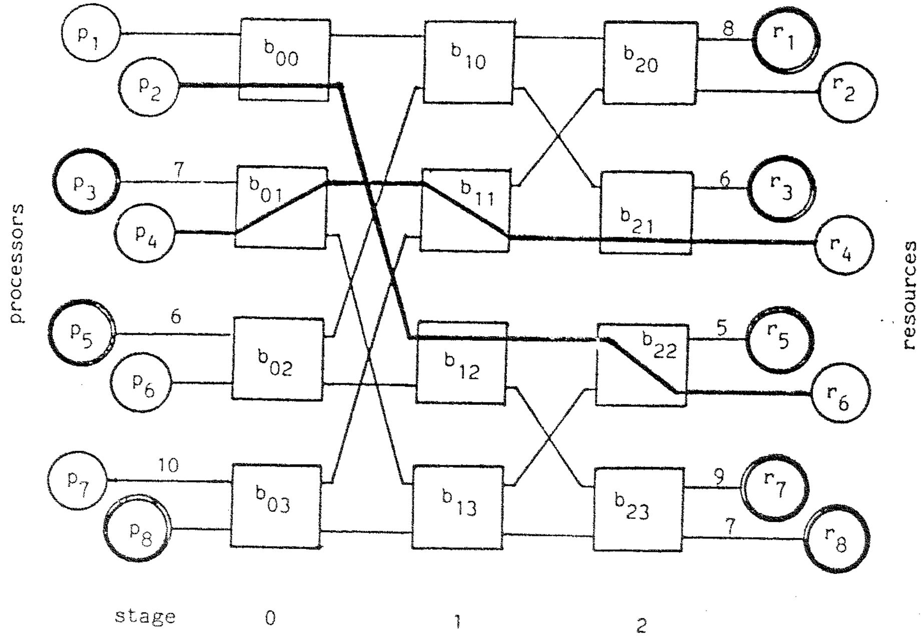


Figure 6.6 A MRSIN with request priority and resource preference

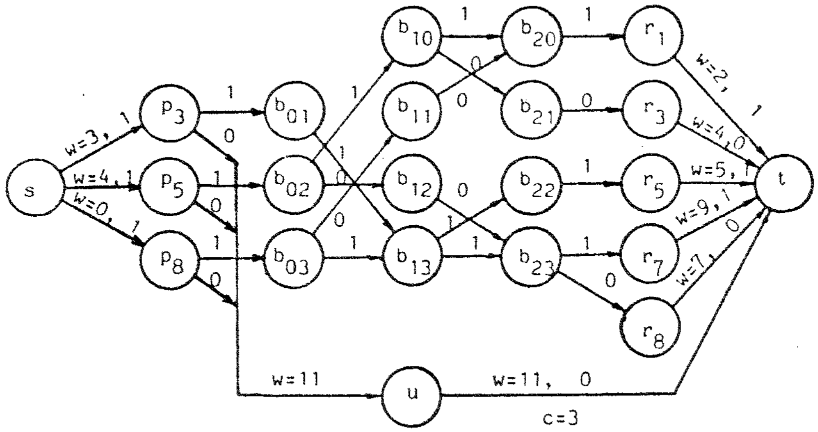


Figure 6.7 The associated flow network obtained from Figure 6.6 by Transfo mation 6.'

Theorem 6.3: The optimal resource mapping on a homogeneous MRSIN with request priority and resource preference is equivalent to the minimum-cost integral flow in the flow network obtained by Transformation 6.2.

<Proof>

It is easy to verify that a feasible flow does exist since one can always push the required amount of flow F_0 through the bypass node u. A flow that passes through the bypass node implies that the corresponding request is not allocated. Thus, maximizing the resource mapping is equivalent to assigning as much flow as possible to the part of the flow network other than the bypass node and is achieved by minimizing the cost of flow assignment. This can be proved by contradiction. Assuming that the minimum-cost flow assignment does not define a maximum allocation, then there exists an s-t path such that the bypass node u is not on this path, and the path is not saturated because at least one unit of flow can advance through it. The remaining flow that could have passed through this s-t path will pass through the bypass node u. According to the cost function w defined in Transformation 6.2, the cost of advancing all the flow through this s-t path is less than that of advancing the same amount of flow through a path passing through node u. The total cost could be reduced if more flow is pushed through the s-t path instead of passing it through node u. The existence of such a path implies that the original assignment is not minimum, which contradicts the assumption. □

Edmonds and Karp have shown a scaled out-of-kilter algorithm to solve the minimum-cost flow problem for a general flow network in polynominal time

[FUL61,EDM72]. For a flow network of 0-1 capacity, the time complexity i bounded by $O(|\mathbf{V}||\mathbf{E}|^2)$. Furthermore, the minimum-cost flow for a flow net work of integral capacity is integral. Thus the optimal resource mapping o homogeneous MRSINs with request priorities and resource preferences can b obtained efficiently. As an example, applying the minimum-cost flow algorithn on the flow network in Figure 6.7 results in the following processor-resourc mappings: $\{(p_1,r_5), (p_5,r_1), (p_8,r_7)\}$. The flow values of the arcs are also show in Figure 6.7.

6.4. Optimal Resource Scheduling for Heterogeneous MRSINs

A heterogeneous MRSIN consists of resources of multiple types, and a prc cessor may generate a request of any type. A resource-allocation problem on heterogeneous MRSIN is equivalent to the *multicommodity maximum-flow prol lem*. The necessary transformation is similar to Transformation 6.1 except tha multiple source-sink pairs are introduced.

For k types of resources in the MRSIN, a multicommodity flow networ has k source-sink pairs, (s^i,t^i), $1 \leq i \leq k$. Let F^i be the flow of the i'th commc dity. The search for the maximum flow can be formulated as a linear progran ming problem [AS78]:

Multicommodity Maximum-Flow Problem

Maximize $\sum_{i=1}^{k} F^i$

subject to:

$$(1)\quad \sum_{e\in\alpha(v)} f^i(e) - \sum_{e\in\beta(v)} f^i(e) = \begin{cases} -F^i & v = s^i \\ F^i & v = t^i \quad \text{(flow conservation)} \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, k$

$$(2)\quad 0 \le \sum_{i=1}^{k} f^i(e) \le c(e) \qquad \text{for all } e \in E \qquad \text{(capacity limitation)}$$

The first constraint above indicates that every types of commodity should obey the flow conservation law and no intermediate node will create or absorb commodities of any type. The second constraint asserts that the sum of all the commodities flowing through a link should not exceed the capacity of the link.

A multicommodity flow network of k commodities may be regarded as the superposition of k single-commodity flow networks. Each layer in the superposition represents a single-commodity flow network. The problem of finding the maximum integral flow in a general multicommodity flow network has been shown to be NP-hard [GAR79]. If the constraints of a flow network can be represented by a totally unimodal matrix [AS78], then the maximum-real-flow of the network is an integer [EVA78], and the algorithms developed for determining the maximum-real-flow may be used to obtain the maximum-integer-flow. In other words, an integer programming problem with the prescribed constraints can be solved as a linear programming problem. The *Simplex Method* is a simple yet efficient method for solving linear programming problems, and has been shown empirically to be a linear-time algorithm [MCC82]. Most multistage interconnection networks, including the cube and Omega, have transformations that belong to this class.

The optimal mapping on a heterogeneous MRSIN with request prioritie and resource preferences can also be obtained by transforming the problem int a multicommodity minimum-cost flow problem. Let $w^i(e)$ be the cost per uni flow for the i'th commodity on edge e, and $f^i(e)$ be the corresponding flow. multicommodity minimum-cost flow problem may be formulated as follows:

Multicommodity Minimum-Cost Flow Problem

Minimize $\sum_{i=1}^{k} \sum_{e\in E} w^i(e) f^i(e)$

subject to:

$$(1)\quad \sum_{e\in\alpha(v)} f^i(e) - \sum_{e\in\beta(v)} f^i(e) = \begin{cases} -F_0^i & v = s^i \\ F_0^i & v = t^i \quad \text{(flow conservation)} \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, k$

$$(2)\quad 0 \le \sum_{i=1}^{k} f^i(e) \le c(e) \qquad \text{for all } e \in E \qquad \text{(capacity limitation)}$$

F_0^i in the flow-conservation constraint is a predetermined flow assignment the link (t^i, s^i), and means that a fixed amount of the i'th-type commoditi have to be circulated from sink t^i to source s^i. The equivalent flow networ consists of k source-sink pairs and k bypass nodes, where k is the number resource types being requested. As in the case with no request priority, t flow network may also be regarded as the superposition of k single-commodi flow networks, and Transformation 6.2 can be applied to each layer.

The multicommodity minimum-cost flow problem for a heterogeneo MRSIN may be solved in a similar way as in solving multicommodi maximum-flow problem.

6.5. A Systolic Architecture with Broadcasting for MRSIN

In this section, a systolic-array architecture for a homogeneous MRSIN with random allocation is examined. The objective is to realize a resource scheduling algorithm on a VLSI chip and to obtain a MRSIN that accomplishes the optimal resource allocation at the speed of signal propagations. In this architecture, the scheduling intelligence is distributed into the switching elements of the network. It is a distributed realization of Dinic's maximum network flow-algorithm.

The corresponding architecture with priorities and preferences based on the out-of-kilter algorithm may be obtained with slight modifications. However, VLSI implementation of the scheduling algorithms for heterogeneous MRSINs is very difficult, and software implementation has to be used.

6.5.1 Dinic's Maximum-Flow Algorithm

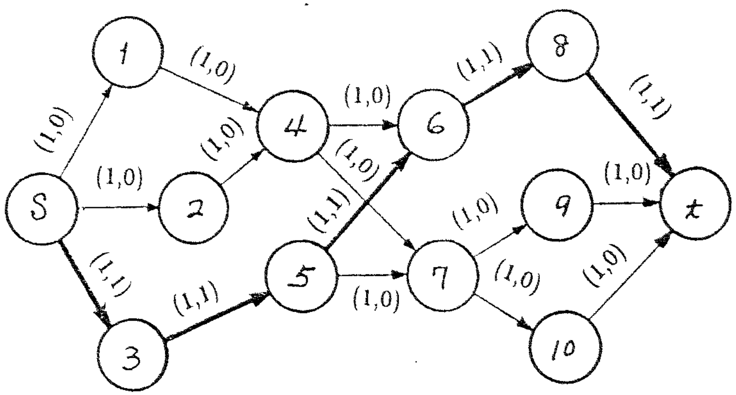
Ford and Fulkerson's maximum network-flow algorithm is a pseudo polynominal-time algorithm. A polynominal-time maximum network-flow algorithm was first proposed by Dinic in 1970 [DIN70]. Dinic's algorithm is also based on the method of augmenting flows iteratively through augmenting paths as described in Section 6.3.2. It is improved over Ford and Fulkerson's algorithm by advancing flows through the shortest augmenting paths. In the Dinic's algorithm, a layered network is used to identify the shortest paths. A

layered network is a weighted digraph which can be derived from a flow net work, G(V,E), with a flow assignment f using the following procedure:

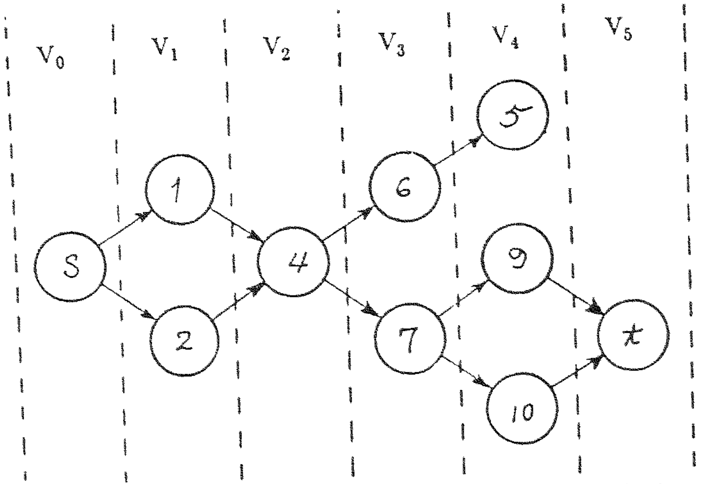
procedure 6.1: Construction of a layered network from G(V,E) with flo f

```
(1) $V_0 \leftarrow \{s\}$, $i \leftarrow 0$.
(2) Construct $T \leftarrow \{v \mid v \notin V_j$ for $j \leq i$ and there is a useful link from
    vertex of $V_i$ to $v\}$.
(3) If T contains t, then $\ell \leftarrow i+1$, $V_\ell \leftarrow \{t\}$, and return.
(4) If T is empty, then no more augmenting path exits, and the prese
    mapping is maximum.
(5) Let $V_{i+1} \leftarrow T$, $i \leftarrow i+1$, and goto step (2).
```

A *useful link* is a link through which more flow may be advanced by either ca celing existing flow or by adding more flow to it. In a layered network, t vertices of the original network are arranged into disjoint sets, $V_0, V_1, \cdots, V$ such that no link points from V_j to V_i ($i \leq j$). A feasible flow in a layered n work is said to be maximal if every (s,t)-directed path in the layered network saturated. Note that a maximal flow needs not be a maximum flow. Comp ing the maximal flow is easier than computing the maximum flow since canc ing flow in reverse links does not have to be considered. The maximal f obtained in the layered network is a net increment to the existing flow. Si the maximum flow is finite, it can be obtained in a finite number of steps us the layered-network construction. An example illustrating the construction a layered network is shown in Figures 6.8a and 6.8b.



(a) A flow network with a flow advaning through the path s-3-5-6-8-t.



(b) The layered network constructed from flow network shown in (a). The maximal flow of this netwok advances through the path s-1-4-7-9-t.

Figure 6.8 Illustration of the generation a layered network

6.5.2 A Systolic MRSIN

The systolic architecture [KUN81] proposed in this section is embedded in the multistage interconnection network. The scheduling intelligence is distributed into the switching elements, resources, and request generators. A processor is connected to the network through a request server (RQ), and a resource is monitored by a resource server (RS). A common bus links these components together. The block diagram of this architecture is illustrated in Figure 6.9 by an 8-by-8 Omega network. Autonomous processes in each component communicate with each other by status signals via the bus and tokens via direct links. The signals on the status bus are the logical OR of the signals set by individual processes, which are accessible by every element in the system.

The network switching elements are responsible for the construction o layered networks to find the augmenting paths, while the other components ar responsible for the synchronization of the phase transitions in the iterations o the scheduling algorithm. RQ accepts a request from the processor and broad casts a request-pending status to the bus. Whenever a resource becomes ready the associated RS will send a resource token to the network that will eventu ally arrive at the requesting RQs. A requesting RQ will submit a request toke to the network to bid for a resource as soon as it receives a resource toke When a request token is received by a ready RS, it will acknowledge th request by returning an acknowledgement token. The RQ and RS will form matched pair, and the path connecting them is registered until it is allocated the end of the current scheduling phase or it is canceled by possible re-pairing

To achieve optimal mapping, a switching process has to follow sever token-propagation rules that emulate the procedures of the constructions

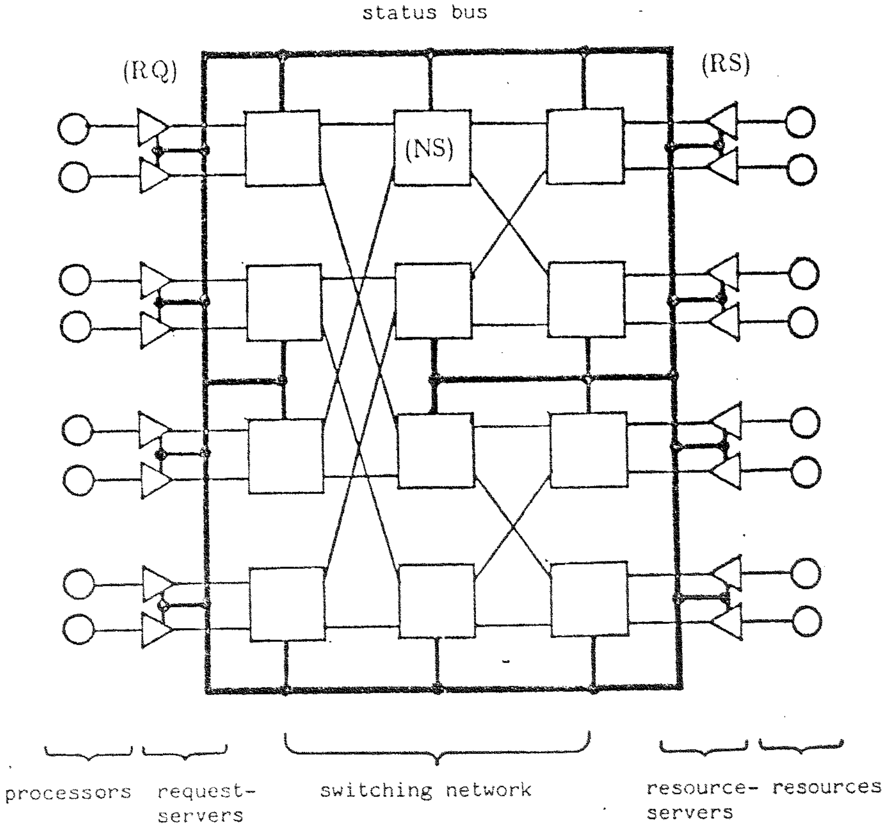


Figure 6.9 A systolic architecture of a homogeneous MRSIN embedded in an 8×8 Omega network

layered networks and finding the maximal flow in a layered network. If a patl is free, the corresponding switching process will deliver any resource toke received from the resource side to the processor side. These directions ar reversed if the path is registered. A request token is expected from where resource token was delivered. In case that there are more than one sendin path, a resource token is duplicated, and one token is sent along each patl No duplication is done for the request token, and one of the paths is chosen a random.

To avoid chaos in the parallel search of augmenting paths, the schedulin procedure is divided into alternating phases. In each phase, only one type tokens are allowed to propagate. The phase transitions is synchronized b broadcasting the status of each process on the status bus. The scheduling pr cess is terminated when all the tokens are blocked in the resource-token distr bution phase.

Based on these rules of token propagation and link reconfiguration, it easy to verify that the resource-token propagation rules are essentially a para lel construction of the layered network, and request-token propagations a equivalent to finding the maximal flow in the lyered network. The conditic under which no request-token is able to reach a ready RS is equivalent to tl condition when every (s,t)-directed path is saturated. Therefore, whenever tl iterations of the bidding and acknowledgement phases terminate, the maxim mapping is achieved.

The synchronization bus has five signal lines. Let s_i's be the boole representation of each signal on the bus. Their values are interpreted as f lows:

$$s_1 = \begin{cases} 1 & \text{at least one processor generates a request,} \\ 0 & \text{otherwise.} \end{cases}$$

$$s_2 = \begin{cases} 1 & \text{at least one resource is ready,} \\ 0 & \text{otherwise.} \end{cases}$$

$$s_3 = \begin{cases} 1 & \text{at least one request token is propagating,} \\ 0 & \text{otherwise.} \end{cases}$$

$$s_4 = \begin{cases} 1 & \text{at least one acknowledgement token is propagating,} \\ 0 & \text{otherwise.} \end{cases}$$

$$s_5 = \begin{cases} 1 & \text{at least one resource token is propagating,} \\ 0 & \text{otherwise.} \end{cases}$$

In terms of these five status signals, the state is defined by a five tuple (s_1,s_2,s_3,s_4,s_5). The state transitions of the MRSIN are driven by processes RQ's, RS's, and NS's that set or reset signal registers in each switching element. The system is driven into state (1,1,0,0,0) when there are pending requests and ready resources. Once entering this state, the ready RQs start to distribute their resource-tokens and drive the MRSIN into state (1,1,0,0,1). While in this state, resource-tokens are propagating through the network until either of the following events happens: (1) all tokens are blocked, or (2) any of the requesting RQs receives a token. In the former case, no mapping is possible, and the MRSIN goes back to the previous state. In the latter case, the requesting RQs will search for ready resources by propagating request-tokens and setting bus signals. In response to request signals, all NSs stop distributing resource-tokens and prepare to propagate request tokens in the reverse direction. The MRSIN is thus driven into state (1,1,1,0,0) by the RQs. Now, the ready RS's are waiting for the resource tokens. Should a request token reach a ready RS, the RS replies by setting the acknowledgement signal on the bus to drive the system into state (1,1,0,1,0) and sending an acknowledgement-token (or ack-token in short) into the network. Ack-tokens will be relayed to requesting RQ's. The RQ's receiving an ack-token is allocated a resource. Those unacknowledged RQs will try to bid for resources in the next iteration unless all request-tokens are blocked, and the MRSIN goes back to state (1,1,0,0,0). A new iteration is initiated by the remaining ready RS's and the MRSIN goes back to state (1,1,0,0,1). Should all ready-tokens be blocked, no further increase in resource mapping is possible, and the MRSIN enters the allocation state to allocate those registered RQ-RS pairs. These RQs and RSs have to cancel their request-pending and resource-ready signals. Some RQs and RSs may remain unallocated, so the MRSIN may enter one of the following states (0,0,0,0,0), (1,0,0,0,0), (0,1,0,0,0), and (1,1,0,0,0). A state transition diagram of the MRSIN is illustrated in Figure 6.10. The control flow of this transition diagram is the same as that of Dinic's algorithm. It is easy to verify that whenever the MRSIN comes out from the loop, (1,1,0,0,1) → (1,1,1,0,0) → (1,1,0,1,0) → (1,1,0,0,0) → (1,1,0,0,1), the optimal mapping is obtained.

6.5.3 The Control Processes in A Network Switching Element

Processes RQ's and RS's may be realized by simple finite-state sequential machines. In this section, we show the design of the switching element by a finite-state machine. The following design is developed with respect to a MRSIN network with 2-by-2 crossbar switches. Each switch can be in a parallel connection or in a cross connection. The setting of the connections

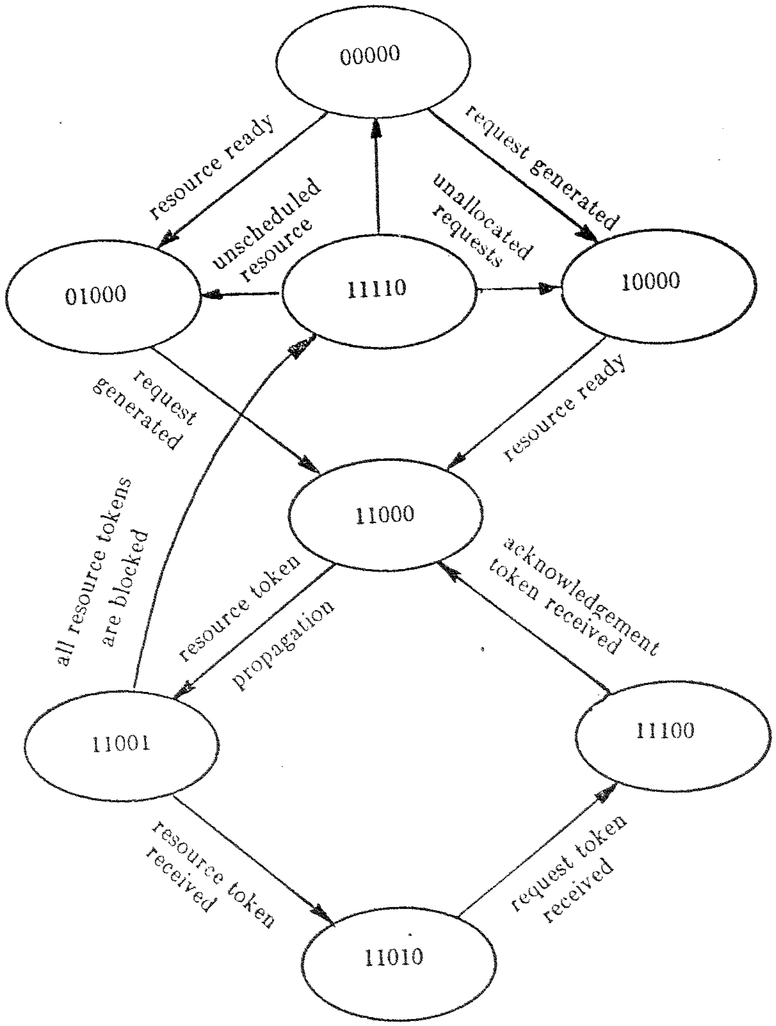


Figure 6.10 State transition diagram of a homogeneous MRSIN without priority and preference

determines the two paths passing through the switch. For convenience, the paths are labeled as shown in Figure 6.11. The status of a path may be free, registered, or occupied. It is occupied if the path has already been allocated t a processor-resource pair. While not being occupied, it may be registered o free depending on whether it is temporarily engaged in connecting a processor-resource pair. A registered path is isomorphic to an s-t path of non-zero flow i the equivalent flow network.

In the current design, the configuration of a switch is controlled by a configuration register of five bits. These bits are designated as X, F_1, F_2, G_1, and G_2, and their functions are defined below :

$$X = \begin{cases} 1 & \text{the switch is in a cross connection} \\ 0 & \text{the switch is in a parallel connection} \end{cases}$$

$$F_jG_j = \begin{cases} 11 & \text{the } j'\text{th path is registered} \\ 10 & \text{the } j'\text{th path is free} \\ 0x & \text{the } j'\text{th path is occupied} \end{cases} \qquad \text{for } j=1,2$$

During a scheduling phase, an NS is responsible for relaying tokens. Th configuration of a switch determines the way that tokens are received an passed. Depending on the status of a path, a resource token is expected t come in from the resource-side if the path is free, and is expected to come i from the processor-side if the path is registered. The latter case is equivaler to a flow cancellation. Let the four ports of a switch be labeled as q_1, q_2, r and r_2, in which q indicates the port at the processor side and r indicates tl link at the resource side (Figure 6.11). The port from which a certain type token is expected may be determined by the configuration registers. F

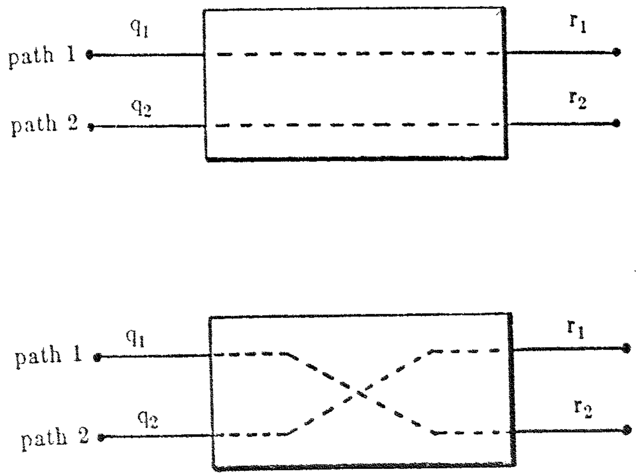


Figure 6.11 Path labeling of a 2 × 2 switch

example, if a resource token is expected from port q_1, then the value of F_1G
must be 11. In the following definitions, the LHS is a concatenation of toke
type and port identification, which represents that the designated type of toke
is expected from the corresponding port if the Boolean expression on the RH
is true.

$$\text{resource token}-q_1 = F_1G_1$$

$$\text{resource token}-q_2 = F_2G_2$$

$$\text{resource token}-r_1 = \overline{X}F_1\overline{G_1} + XF_2\overline{G_2}$$

$$\text{resource token}-r_2 = XF_1\overline{G_1} + \overline{X}F_2\overline{G_2}$$

$$\text{request token}-q_1 = F_1\overline{G_1}$$

$$\text{request token}-q_2 = F_2\overline{G_2}$$

$$\text{request token}-r_1 = \overline{X}F_1G_1 + XF_2G_2$$

$$\text{request token}-r_2 = XF_1G_1 + \overline{X}F_2G_2$$

With this set of Boolean expressions, a switch can determine from which por
given type of tokens may be received and to which port a token shall be sent

The resource tokens are sent to ports where request tokens are expect
and a request token is passed to the port from which a resource token
received. Whenever a ready token is received, a switch duplicates as m
tokens as it has to send and rejects any resource token received late. On
other hand, a request token is not duplicated at any switching element. I
simply passed over to the NS in the next stage. In passing request tokens, th

may be only one request token accepted but have two possible ports to send, or two request-token accepted and only one port to send. In either case, a random choice is made to determine a unique port. These token passing rules can be implemented by simple logic circuits.

To mark whether a token has been received from any port, four one-bit token-marker registers are introduced and designated as Q_1, Q_2, R_1, and R_2. They may be referred as vectors $\vec{Q}$ and $\vec{R}$ whenever it is convenient to do so. A block diagram of the organization of a switching element is shown in Figure 6.12.

A switch is reconfigurated after every iteration of the layered-network construction and finding maximal flow. During a reconfiguration, the value of a configuration register is changed depending on the status of the corresponding path. If a link passes both a request token and an acknowledgement token, it will be registered during a reconfiguration. Two simple rules may be applied: (1) registering a free link yields a registered one, and (2) registering a registered link results in a free one. Based on these rules, all possible reconfigurations of a switch initially in configuration $(X,F_1,F_2,G_1,G_2) = (1,1,1,1,0)$, is shown in Figure 6.13. A configuration is represented symbolically in this figure, in which two paths of a switch are indicated. The status of a path is indicated by three different types of lines, and the condition in which tokens are propagated is indicated by arrows and dots. The digram only shows reconfiguration of a switch in one state. A switch may be in 18 states, and the reconfiguration control is rather complicated. An approach to simplify the design of the reconfiguration control logic is discussed next.

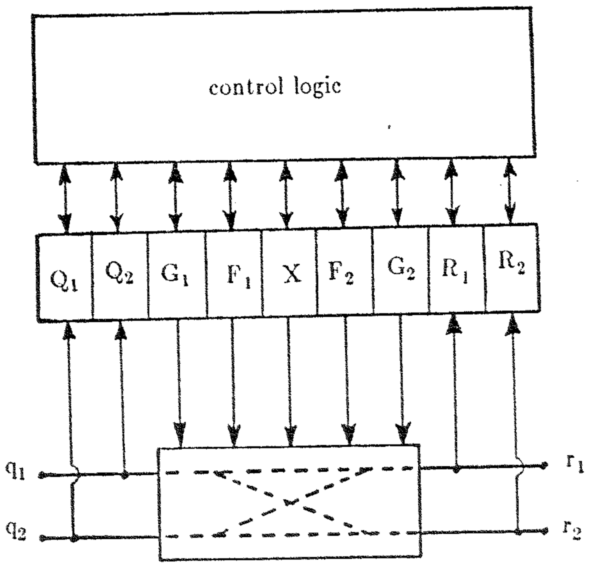


Figure 6.12 Organization of a switching element

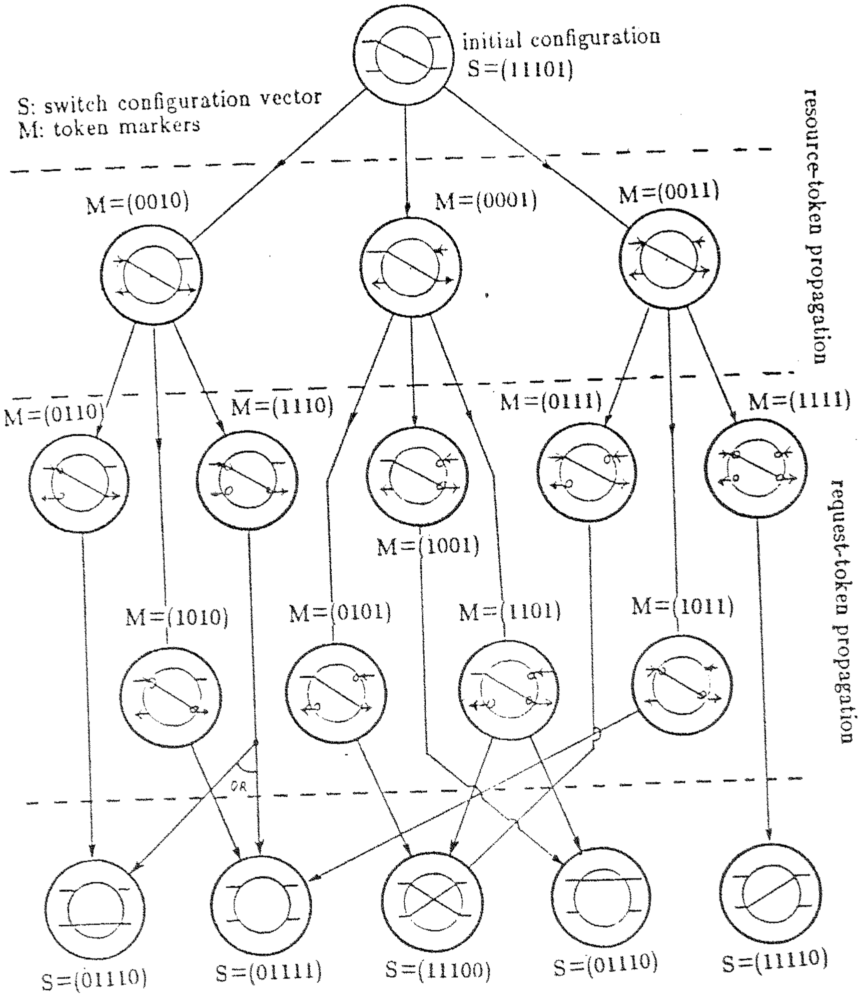


Figure 6.13 Partial state transition diagram of a switching element

Let the status of a configuration register prior to a reconfiguration be attributed by an index t and that after reconfiguration be attributed by t+1. Then the reconfiguration control of a switch may be represented by the Boolean equations B_X and B_G. They are the functions of the current configurations.

$$X(t+1) = B_X(X(t),\vec{F}(t),\vec{G}(t),\vec{Q}(t),\vec{R}(t)) \quad (6.1a)$$

$$\vec{G}(t+1) = B_G(X(t),\vec{F}(t),\vec{G}(t),\vec{Q}(t),\vec{R}(t)) \quad (6.1b)$$

Based on these equations, the issue of designing a reconfiguration control may be derived independently. The derivation of B_X is discussed first. The main idea behind the derivation is to count the conditions that cause changes o status. This may be simplified by defining two Boolean variables α_X and β_X a follows:

$$\alpha_X = \begin{cases} \text{true} & \text{if X changes from 1 to 0} \\ \text{false} & \text{otherwise} \end{cases}$$

$$\beta_X = \begin{cases} \text{true} & \text{if X changes from 0 to 1} \\ \text{false} & \text{otherwise} \end{cases}$$

B_X may be expressed as in Eq. 6.2

$$B_X = X(t)\overline{\alpha_X} + \overline{X(t)}\beta_X \quad (6.$$

The variables α_X and β_X may be expressed by Boolean functions of the state configuration registers and token-marker registers. Based on the lin reconfiguration rules stated above, the conditions under which the swit configuration will change from a cross connection to a parallel connection a shown exhaustively in Figure 6.14, and those under which the configurati changes from a parallel connection to a cross connection are shown in Figure 6.15. The same symbolic notation as in Figure 6.13 is used in theses figures. By combining the Boolean representations of these conditions and simplifying it yields Eq's. 6.3a and 6.3b as shown below:

Fact 1 :

$$\alpha_X = F_1F_2((G_1 \oplus G_2)(Q_1Q_2 \oplus R_1R_2) + (G_1 \odot G_2)(Q_1R_1 \oplus Q_2R_2)) \qquad (6.3a)$$

$$\beta_X = F_1F_2((G_1 \oplus G_2)(Q_1Q_2 \oplus R_1R_2) + (G_1 \odot G_2)(Q_1R_2 \oplus Q_2R_1)) \qquad (6.3b)$$

Where $\oplus$ and $\odot$ are the logic operators of ***exclusive** OR* and ***inclusive** OR*, respectively. The index t for all Boolean variables in Eq's 6.3a and 6.3b is neglected. Similarly, two Boolean variables associated with the status changes of G_1, α_{G_1}, and β_{G_1}, may be defined as follows:

$$\alpha_{G_j} = \begin{cases} \text{true} & G_j \text{ changes from 1 to 0} \\ \text{false} & \text{otherwise} \end{cases} \qquad \text{for } j=1,2.$$

$$\beta_{G_j} = \begin{cases} \text{true} & G_j \text{ changes from 0 to 1} \\ \text{false} & \text{otherwise} \end{cases} \qquad \text{for } j=1,2.$$

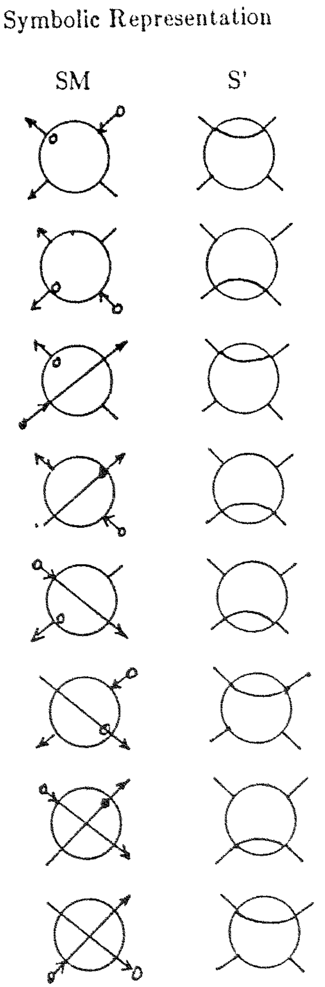
Then,

$$B_{G_j} = G_j(t)\overline{\alpha_{G_j}} + \overline{G_j(t)}\beta_{G_j} \qquad \text{for } j=1,2. \qquad (6.4)$$

Fact 2 :

$$\alpha_{G_1} = (\overline{X}Q_1R_1 + XQ_1R_2)(F_2Q_1(G_2\overline{X}\overline{Q_2}\overline{R_1}R_2 + G_2X\overline{Q_2}R_1\overline{R_2} + \overline{G_2}Q_2\overline{R_1}\overline{R_2}) \qquad (6.5a)$$

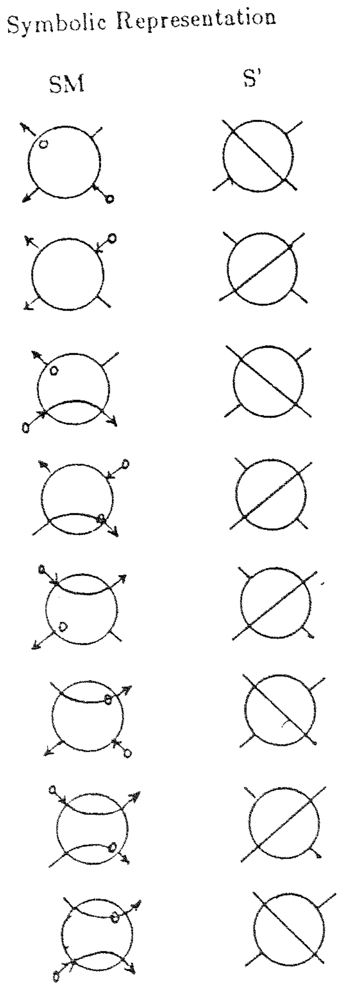
$$\beta_{G_1} = (\overline{X}Q_1R_1 + XQ_1R_2)(F_2Q_1(\overline{G_2}\overline{X}Q_2\overline{R_1}R_2 + \overline{G_2}X\overline{Q_2}R_1\overline{R_2} + G_2Q_2\overline{R_1}\overline{R_2}) \qquad (6.5b)$$



Boolean Representation

S	M	S'
(11100)	(1010)	(01110)
(11100)	(0101)	(01101)
(11101)	(1100)	(01110)
(11101)	(0011)	(01101)
(11100)	(1100)	(01101)
(11110)	(0011)	(01110)
(11111)	(1010)	(01101)
(11111)	(0101)	(01110)

Figure 6.14 All the possible state transitions during a switch reconfigurat from a cross connection to a parallel connection.



Boolean Representation		
S	M	S'
(01100)	(1001)	(11110)
(01100)	(0110)	(11101)
(01101)	(1100)	(11110)
(01101)	(0011)	(11101)
(01110)	(1100)	(11110)
(01110)	(0011)	(11101)
(01111)	(1001)	(11110)
(01111)	(0110)	(11101)

Figure 6.15 All the possible state transitions during a switch reconfiguration from a parallel connection to a cross connection

The index t for all Boolean variables in Eq's 6.5a and 6.5b is also neglected The two expressions are also obtained by logic ORing of all possible condition that will cause a status change of path 1. These conditions are shown in Fig ures 6.16 and 6.17. Because of the symmetry between path 1 and path 2, th Boolean expressions of α_{G_2} and β_{G_2} may be obtained by a change of variabl from Eq's 6.5a and 6.5b as follows :

Fact 3 :

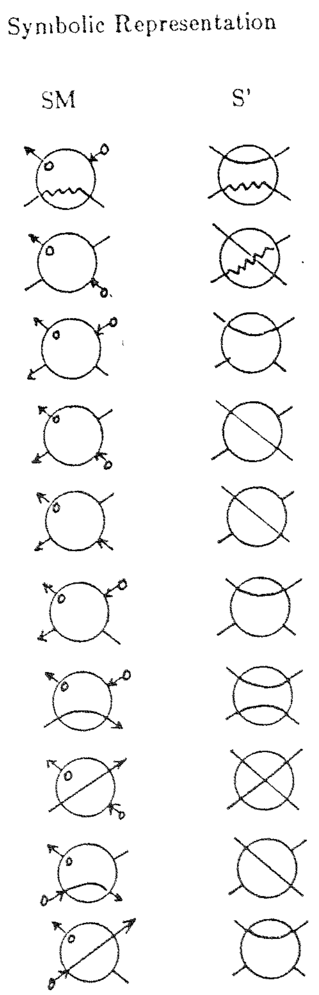
$$\alpha_{G_2}(X,F_1,F_2,G_1,G_2,Q_1,Q_2,R_1,R_2) = \alpha_{G_1}(X,F_2,F_1,G_2,G_1,Q_2,Q_1,R_2,R_1) \quad (6.6$$

$$\beta_{G_2}(X,F_1,F_2,G_1,G_2,Q_1,Q_2,R_1,R_2) = \beta_{G_1}(X,F_2,F_1,G_2,G_1,Q_2,Q_1,R_2,R_1) \quad (6.6$$

The sequence control of this process is straightforward and the control logic a also relatively simple due to the Boolean functions obtained above. Th results indicate the feasibility of a VLSI implementation.

The states of F_1 and F_2 do not change until their corresponding paths allocated at the end of each scheduling cycle. The controls for the state tra tions are very simple and will not be discussed further.

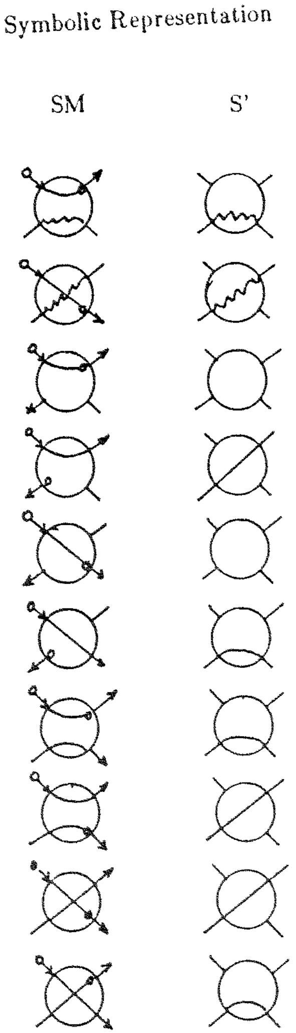
For an a×b switch, there are $\begin{bmatrix}a\\b\end{bmatrix}b!$ (if $a \geq b$) or $\begin{bmatrix}b\\a\end{bmatrix}a!$ (if $a < b$) connect patterns. It requires $\log_2\begin{bmatrix}a\\b\end{bmatrix}b!$ (assuming $a \geq b$) registers to control the c nection of such a switch. Besides, there are up to b paths passing through switch. A pair of registers F_iG_i is necessary for characterizing the statu each path. Hence, the complexity of the control logic for an a×b switch w be substantially higher than those described above.



Boolean Representation

S	M	S'
(01000)	(1x1x)	(01010)
(11000)	(1xx1)	(11010)
(01100)	(1x1x)	(01110)
(01100)	(1001)	(11110)
(11100)	(1xx1)	(11110)
(11100)	(1010)	(01110)
(01101)	(1x1x)	(01111)
(11101)	(1001)	(11111)
(01101)	(1100)	(11110)
(11101)	(1100)	(01110)

Figure 6.16 All the possible state transitions during a switch reconfiguration (path 1 was registered and becomes free)



Boolean Representation

S	M	S'
(01010)	(1x1x)	(01000)
(11010)	(1x1x)	(11000)
(01110)	(1x1x)	(01100)
(01110)	(1100)	(11101)
(11110)	(1001)	(11100)
(11110)	(1100)	(01101)
(01111)	(1010)	(01101)
(01111)	(1001)	(11101)
(11111)	(1001)	(11101)
(11111)	(1010)	(01101)

ɡure 6.17 All the possible state transitions during a switch reconfiguratio (path 1 was free and becomes registered)

6.6 Concluding Remarks

In this chapter, we have transformed the various resource mapping problems into network-flow problems for which efficient algorithms exist. For MRSINs with homogeneous resources, efficient distributed scheduling algorithms have been developed. An efficient VLSI implementation for the distributed algorithm with random allocations can is also shown. With the proposed VLSI systolic-array architecture, the augmenting paths are searched in parallel, and the time complexity is measured in terms of gate delays instead of instruction execution cycles. Therefore, the scheduling algorithm will run at least 100 times faster than a software implementation. For MRSINs with heterogeneous resources, no distributed algorithm has been found and software implementation of the centralized scheduling algorithm is necessary. Table 6.1 is a summary of the results obtained regarding to the resource allocation on MRSINs.

The transformation methods described in this chapter is simple, yet they are not limited to solving resource mapping problems only. They can be applied to solve load balancing problems and packet-routing problems in conventional multistage interconnection networks. The former problem can be transformed to the minimum-cost flow problem by encoding processor loads into request priorities and resource preferences. The latter problem requires a packet to be sent through a route of the shortest delay and arrive at a processor with the minimum response time. This is achievable by encoding buffer delays into link costs in the minimum-cost flow problem.

Table 6.1 Summary of the proposed distributed resource allocation scheme on multistage resource sharing interconnection networks

Scheduling		*Homogeneous*		*Heterogeneous*	
Discipline		Random	P & P	General	Multistage
Optimal Scheduling	Equivalent Flow Problem	Max-Flow	Min-Cost Circulation	Integer Multi-Commodity	Real Multi-Commodity
	Algorithm	Ford-Fulkerson, Dinic	Out-of-Kilter	NP Hard	Linear Programming
Distributed Algorithm		Yes	Yes	NA	NA
Implementation		Distributed Architecture Synchronized by Broadcasting Communicate by Token Propagation	Software	NA	Software

CHAPTER VII
SUMMARY AND CONCLUSIONS

The results obtained in this thesis and the new research topics to be investigated are summarized in this chapter. We will also discuss the promising applications that the algorithms designed in this study can be extended.

7.1 Summary

In this thesis, we have studied the resource-allocation problem in resource sharing computer systems. A resource sharing system is characterized by a pool of request generators and a pool of resources interconnected by a resource sharing interconnection network. Central issues in resource scheduling include the minimization of resource conflicts, the reduction of the probability of network blockage (or packet congestions), and the balance of workload among resources. Evaluations indicated that distributed state-dependent scheduling schemes are preferable. Moreover, integrating the scheduling schemes into the network protocol significantly reduces the overhead of collecting status information and enhances the speed of scheduling.

A methodology has been proposed to optimize the resource mapping, reduce the scheduling overhead, and facilitate a fast implementation. The

methodology has been applied to the design of resource-allocation schemes fo
three representative networks of increasing complexities.

For a single contention-bus network, the resource scheduling problem i
reduced to the problem of identifying a station with the extremum paramete
among a set of physically dispersed random numbers. A distribute
minimum-search algorithm that utilizes the collision-detection capability of th
contention bus has been proposed. The window-search procedure can resolv
the global minimum in an average of 2.4 contention steps. No explicit messag
transfer is required in this process.

For a multiple contention-bus network, the resource-allocation problem i
reduced to the problem of ordered selections. A multi-window parallel searc
procedure that is an extension of the single-bus search procedure has been pr
posed to select the processors with minimum numbers. The average time con
plexity of this search procedure is about O($\log_2 t$), where t is the number
buses and is equal to the number of processors to be selected. The mappin
between the selected resources and processors has been found to be the classic
stable-marriage problem.

For resource allocation on multistage interconnection networks, the pro
lem is transformed into different network-flow optimization problems, for whi
there exists many efficient algorithms. For systems with homogeneous reque
and resources, the network-flow algorithm has been integrated into the netwo
protocol with a VLSI systolic-array architecture that allows resource scheduli
to be carried out at signal propagation speed.

7.2 Extended Applications

The algorithms developed in this thesis research may be considered as alternative solutions to many problems identified in other areas. Some examples are briefed in this section.

7.2.1 The Distributed Minimum-Search Algorithm

The proposed window search procedure is basically a contention-resolution algorithm that resolves the distributed bus-access problem of CSMA-type networks [BER84, CAP79a, GAL78, HLU81, TOW84]. It performs much better than the Binary-Exponential-Backoff algorithm of the notable Ethernet [MET76]. It also provides a unified framework for optimizing many existing adaptive CSMA protocols including the Urn protocol [KLE78, MIT81], Arrival-Time-Window protocol [GAL78, MOS82a-b, MOS85, KUR83, KUR84, TOW82], and Adaptive-Tree-Walk protocol [CAP79a, CAP79b]. A detailed treatment to this subject is given in Appendix B.

The access of CSMA networks with priority has been studied extensively [CHL80, NI83, TOB82, SHA83, GOL83, ROM81]. The proposed window-search procedure is an elegant solution to this problem and is also discussed in Appendix B.

The distributed query processing is crucial to the performance of distributed databases. Wah and Lien have discovered that the proposed minimum-search algorithm leads to an efficient heuristic solution to this problem [WAH85]. Finding the extremum in a distributed environment by itself is also an interesting problem [BOK81, BOK84, DOL82 SHI81].

7.2.2 The Multi-Window Ordered-Selection Algorithm

The ordered-selection problem in a distributed environment has been investigated intensively in recent years [BAT68, PRE78, KRU82, SHI81, THO77, WAH84a]. Many special architectures have been proposed. Notable among them is a lower-bound architecture proposed by Prepareta which is able to solve an ordered-selection problem with worst case time complexity of O($\log_2$n), where n is the number of distributed random numbers [PRE78, AIG82, BLU73, YAO80, WAH84a]. The multi-window ordered-selection algorithm can also carry out the same task with an average time complexity of O($\log_2$t), When implemented on a multiple contention-bus network of t channels. Thus, it is a practical alternative solution to the selection problem.

7.2.3 The Systolic Max-Flow Architecture

Network flow optimization algorithms are recognized as an important modeling tool in many disciplines. Many efficient algorithms have been developed. Distributed version of these algorithms have also been studied [CHE83, SEG82]. However, the synchronization of distributed processes in these algorithms relies on message passing that is a significant overhead in the system. The systolic architecture with a broadcast bus proposed in this thesis can be used to synchronize distributed processes for maximum-flow computations. The processes in the proposed systolic architecture are synchronized by a broadcast bus. This approach eliminates the need of message passing and speeds up the algorithm significantly.

7.3 Suggested Future Studies

Although we have obtained some significant results in this thesis research, there remains future problems to be studied. Relevant topics are identified and summarized below.

7.3.1 Fault-Tolerant Resource-Allocation Schemes

Throughout this thesis research, the networks in question are assumed to be free from noise or malfunctions of its components. The proposed resource scheduling schemes may fail when inconsistent status information is received by different stations due to noise or transient errors. To design a robust resource scheduling scheme is an important issue to be studied.

7.3.2 Adaptation to Network Operating Systems

The conventional operating-system design considered the network as a message passing mechanism only. All high-level applications are built on top of this mechanism. This thesis suggests that the resource scheduling tasks may be integrated into the underlying network protocol. This approach would have great impacts on the design of network operating systems. Future network operating systems should also include special primitives for supporting other high-level applications.

7.3.3 Special Purpose Computer Networks

Research on interconnecting different families of computer systems with diverse applications is important. This focus results in a flexible but inefficient

communication protocol. Future studies on a flexible and efficient resourc sharing system for many diversified heterogeneous resources is essential. Thus it is desirable to have a network which may be customized to special applications.

7.3.4 Distributed Algorithms Synchronized by Message Broadcasting

Most existing algorithms for interprocess communication rely on point-to point message passing. As indicated in this thesis research and other researcl studies, the synchronization with message broadcasting outperforms its messag passing counterpart. Further, broadcast networks such as Ethernet are popu lar. The future design of distributed algorithms should utilize the broadcas capability of networks.

7.4 Conclusions

This research spans across various disciplines in computer engineerin including computer architectures, computer communications, operating sys tems, and algorithm design. By successfully distributing the scheduling intelli gence into the network protocol, we have developed efficient schemes for sup porting distributed state-dependent resource scheduling that includes loa balancing and resource sharing. The proposed extremum-search scheme is als a realizable optimal backoff algorithm for CSMA networks and provides unified optimization model for the family of adaptive CSMA protocols. Th primitives derived simplify several difficult problems in distributed databas on local computer networks, such as concurrency control and distributed query processing. Better solutions to the ordered-selection and maximum-flow problems have also been developed. The study on multiple contention-bus networks provides insights into the routing control of multi-channel broadband communication networks. The research also generates many new topics for future studies.

LIST OF REFERENCES

[AND83] G. R. Andrew, and F. B. Schneider, "Concepts and Notations for Concurrent programming," *ACM, Computing Surveys, Vol. 15,* March 1983, pp. 3-43.

[ARR81] K. Arrow, L. Pesotchinsky, and M. Sobel, "On Partitioning A Sample With Binary-Type Questions in Lieu of Collecting Observations," *Journal of the American Statistical Association*, Vol. 76, No. 374, June 1981, pp. 402-409.

[AIG82] M. Aigner, "Parallel Complexity of Sorting Problems," *Journal o Algorithms 3,* Academic Press, Inc., 1982, pp. 79-88.

[AS78] A.A. Assad, "Multicommodity Network Flows - A Survey," *Net works*, Vol. 8, 1978, pp. 37-91

[BAE80] J. Baer, *Computer Systems Architecture,* Computer Science Press Inc., Rockville, Maryland, 1980.

[BAT68] K.E. Batcher, "Sorting Network and Their Applications," *Proc 1968 Spring Joint Computer Conference,* AFIPS press, Vol. 32, 1968 pp. 307-314.

[BER84] T. Berger, N. Mehravari, D. Towsley, and J. Wolf, "Randon Multiple-Access Communication and Group Testing," *IEEE Trans on Communications, Vol. COM-32, No. 7,* July 1984, pp.769-778.

[BLU73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjar "Time Bounds for Selection," *Journal of Computer and System sc ence,* No. 1, 1973, pp. 448-461.

[BOK81] S. H. Bokhari, "MAX : An Algorithm for Finding Maximum in A Array Processor With Global Bus," *Proceedings of the 1981 Interna tional conf. on Parallel Processing,* 1981, pp. 302-303.

[BOK84] S. H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus," *IEEE trans. on Computers, Vol. C-33, No. 2,* February 1984, pp. 133-139.

[BON81] J.A. Bondy and U.S.R. Murty, *Graph Theory With Applications,* North-Holland, New York, N.Y., 1981

[BRI82] F.A. Briggs, K.S. Fu, K. Hwang and B.W. Wah, "PUMPS Architecture for Pattern Analysis and Image Database Management," *IEEE Trans. on Comp.*, Vol. C-31, No. 10, Oct. 1982, pp 969-983.

[BRO84] R. L. Brown, P. J. Denning, and W. F. Tichy, "Advanced Operating Systems," *IEEE Computer, Vol. 17, No. 10,* October 1984, pp. 173-190.

[CAP79a] J. Capetanakis, "Tree Algorithm for Packet Broadcast Channels," *IEEE Trans. Information,* Vol. IT-25, No. 5, Sept. 1979, pp.505-515.

[CAP79b] J. Capetanakis, "Generalized TDMA: The Multi-Accessing Tree Protocol," *IEEE Trans. on Communications, Vol. COM-27,* Oct. 1979, pp. 1479-1484

[CDC83] Control Data Corporation, "Technical Information: Control Data Cyberplus," News Release and Fact Sheet, Oct. 1983.

[CHL80] I. Chlamtac, and W.R. Franta, "Message-Basd Priority Access to Local Networks," *Computer Communications,* Vol. 3, No. 2, April 1980, pp. 77-84.

[CHE83] T. Y. Cheung, "Graph Traversal Techniques and The Maximum Flow Problem in Distributed Computation," *IEEE Trans. on Software Engineering,* Vol. SE-9, No. 4, July 1983, pp. 504-512.

[CHO79] Y.C. Chow and W. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. on Comp.*, Vol. C-28, No. 5, May 1979, pp. 334-361.

[CHO82] T.C.K. Chou, and J.A. Abraham, "Load balancing in distributed systems," *IEEE Trans. Software Eng.*, Vol. SE-8, No. 7, July, 1980, pp.401-412.

[CHU80] W. Chu, L.J. Holloway, M.T. Lan, and K. Efe, "Task allocation distributed data processing," *IEEE Computer,* Vol. 13, No. 11, N 1980, pp. 57-69.

[CRU82] R. Cruz, and B. Hajek, "A New Upper Bound to the Through of a Multi-Access Broadcast Channel," *IEEE Trans. on Informat Theory, Vol. IT-28, No. 3,* May 1982.

[DAV70] H.A. David, *Order Statistics,* John Wiley and Sons, Inc., New Yo 1970.

[DAY79] J. D. Day, "Resource Sharing Protocols," *IEEE Computer,* S tember 1977, pp.47-56.

[DEN80] J. B. Dennis, "Data Flow Supercomputers," *IEEE Computer.*, N 1980, pp. 48-56.

[DIJ68] E. W. Dijkstra, "Cooperating Sequential Processes," *Programn Languages,* F. Genuys, ed., Academic Press, New York, 19 pp.43-112.

[DIN70] E.A. Dinic, "Algorithm for Solution of a Problem of Maximal F in a Network with Power Estimation," *Soviet Math. Dokl.*, Vol. 1970, pp. 1277-1280.

[DOL82] D. Dolev, M. Klawe, and M. Rodeh, "An O(nlogn) Unidirectic Distributed Algorithm for Extrema Finding in a Circle," *Journa Algorithms 3,* Academic Press, Inc., 1982, pp. 245-260

[DUB82] M. Dubois, and F. A. Briggs, "Effects of Chache Coherency in M tiprocessors," *IEEE Trans. on Computers.*, Vol. C-31, No. 11, N 1982.

[EDM72] J. Edmonds and R.M. Karp, "Theoretical Improvements in A rithmic Efficiency for Network Flow Problems," *J. ACM,* Vol. No. 2, April 1972, pp. 248-264.

[EVA78] J.R. Evans and J.J. Jarvis, "Network Topology and Integral M ticommodity Flow Problems," *Networks,* Vol. 18, 1978, pp. 107-1

[ENS77] P. H. Enslow, "Multiprocessor Organization," *ACM, Compu Surveys,* Vol. 9, March 1977, pp.103-129.

[FAY77] G. Fayolle, et al., "Stability and Optimal Control of the Packet Switching Broadcast Channel," *JACM*, Vol. 24, No. 3, July 1977, pp. 375-386.

[FEL71] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 2, John Wiley & Sons, New York, 1971.

[FEN81] T.Y. Feng, "A Survey of Interconnection Networks," *IEEE Computer*, Dec. 1981, pp.12-27.

[FOR62] L.R. Ford and D.R. Fulkerson, *Flow in Networks*, Princeton University Press, Princeton, N.J., 1962.

[FRA82] J. M. Frankovich, "A Bandwidth Analysis of Baseline Networks," *International Conf. on Distributed Computing systems*, Oct. 1982, pp. 572-578.

[FRA80] W. R. Franta, and M. B. Bilodeau, "Analysis of a Prioritized CSMA Protocol Based on Staggered Delays," *Acta Informatica*, 13, 1980, pp. 299-324.

[FUL61] D.R. Fulkerson, "An Out-of-Kilter Method for Minimum Cost Flow Problems," *SIAM J. of Computing*, Vol. 9, No. 1, 1961, pp. 18-27.

[FUL78] S. H. Fuller, and S. P. Harbison, *The C.mmp multiprocessor*, Technical Report, Carneigie-Mellon University, Computer Science Dept., 1978.

[GAL78] R. G. Gallagher, "Conflict Resolution in Random Access Broadcast Networks," *Proc. AFOSR Workshop Communication Theory and Applications*, Sept. 17-20, 1978, pp. 74-76.

[GAR79] M.R. Garey and D.S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, San Francisco, 1979.

[GOL81] B. Golden, M. Ball and L. Bodin, "Current and Future Research Directions in Network Optimization," *Computer and Operations Research*, Vol. 8, 1981, pp. 71-81.

[GOL83] Y.I. Gold, and W.R. Franta, "An Efficient Collision-Free Protocol for Prioritized Access-Control of Cable Radio Channels," *Computer Networks 7*, North-Holland Publishing Company, 1983, pp. 83-98.

[HAJ82] B.E. Hajek, "Information of Partitions with application to Randon Access Communication," *IEEE Trans. on Information Theory*, Vol IT-28, No. 5, September 1982, pp. 691-701.

[HAN78] P. B. Hansen, "Distributed Processes: A concurrent programmin concept," *Communications of ACM, Vol. 21,* Nov. 1978, pp.934-941.

[HAY78] J.P. Hayes, *Computer Architecture and Organization*, McGraw-Hill Inc., New York, 1978.

[HIC82] A. Hicks, *Resource Scheduling on Interconnection Networks*, M.S Thesis, Purdue University, Aug., 1982.

[HLU81] M.G. Hluchyj, *"Multiple access Communication: The Finite Use Population Problem,"* MIT Report, LIDS-TH-1162, Nov. 1981 Cambridge, MA.

[HOA74] C. A. R. Hoare, "Monitor: An Operating System Structure Con cept," *CACM*, Vol. 17, Oct. 1974, pp. 549-557.

[HOA78] C. A. R. Hoare, "Communicating Sequential Processes," *Communi cation of ACM*, Vol. 21, August 1978, pp.666-667.

[HWA80] K. Hwang, and L. M. Ni, "Resource Optimization of A Parall Computer for Multiple Vector Processing," *IEEE Trans. on Con puters, C-29,* September 1980 pp. 831-836.

[HWA81] K. Hwang et al., "A Unix-based Local Computer Network Wit Load Balancing," *IEEE Computer*, Vol. 15, No. 4, April 1982, p 55-66.

[HWA84] K. Hwang, and F. A. Briggs, *Computer Architecture and Parall Processing*, McGraw Hill, 1984.

[JAY82] B. Jayaraman, and R.H. Keller, "Resource Expressions for Applic tive Language," *Proc. of the 1982 International Conference Parallel Processing*, August, 1982, pp. 162-167.

[JUA84a] J.Y. Juang, and B.W. Wah, "Unified Window Protocol for Loc Multiaccess Networks," *Proc. of Third Annual Joint Conference the IEEE Computer and Communications Societies*, San Francisc California, April 1984.

[JUA84b] J.Y. Juang, and B.W. Wah, "A Multi-Access Bus-Arbitration Scheme for VLSI-Densed Distributed Systems," *Proc. National Computer Conference*, AFIPS, Vol. 53, July, 1984,pp. 13-22,

[JUA84c] J.Y. Juang, and B.W. Wah, "Optimal Scheduling Algorithms for Resource Sharing Interconnection Networks," *Proc. IEEE 8-th Int'l Computer Software and Applications Conf.*, November 1984.

[KLE72] Kleinrock, L., *Queueing Theory, Volumn I*, Addison-Wesley, 1972.

[KLE75] L. Kleinrock, and F. A. Tobagi, "Packet Switching in Radio Channels: Part I - Carrier Sense Multiple Access Modes and Their Throughput- delay characteristics," *IEEE Trans. Communications*, Vol. COM-23, Dec. 1975, pp. 1400-1416.

[KLE78] Kleinrock, L. and Y. Yemini, "An Optimal Adaptive Scheme for Multiple Access Broadcast Communication," *Proc. ICC*, 1978, pp. 7.2.1-7.2.5.

[KRU82] C. P. Kruskal, "Results in Parallel Searching, Merging, and Sorting," *1982 International Conference on Parallel Processing*, pp. 196-108.

[KRU83] C. P. Kruskal, and M. Snir, "The performance of Multistage Interconnection Networks for Multiprocessors," *IEEE trans. on Computers*, Vol. C-32, No. 12, December 1983, pp. 1091-1098.

[KUN81] H.T. Kung et al., *VLSI Systems and Computations*, Computer Science Press, Inc., Rockville, Maryland, 1981.

[KUN82] H. T. Hung, "Why Systolic Architectures," *IEEE Computer.*, January 1982, pp. 37-46.

[KUR83] J. F. Kurose, and M. Schwartz, "A Family of Window Protocols for Time Constrained Applications in CSMA Networks," *Proc. IEEE INFOCOM 83*, San Diego, CA, 1983, pp. 405-413.

[KUR84] J. F. Kurose, M. Schwartz, and Y. Yemini, "Multiple-Access Protocols and Time-Constrained Communication," *ACM, Computing Surveys, Vol. 16, No. 1*, March 1984, pp.43-70.

[LAN82] T. Lang, M. Valero, and I. Alegre, "Bandwidth of Crossbar an Multiple-Bus Connections for Multiprocessors," *IEEE trans. o Computers*, Vol. C-31, No. 12, December 1982.

[LAN83] T. Lang, M. Valero, and M. A. Fiol, "Reduction of Connections fc Multibus Organization," *IEEE Transactions on Computers*, Vol. C 32, No. 8, August 1983, pp. 707-716.

[LAW75] D. Lawrie, "Access and Alignment of Data in an Array Processor, *IEEE Trans. on Computers*, Vol. C-24, No. 12, Dec. 1975, pp. 21 255.

[LEI81] Dennis W. Leinbaugh, "High Level Specification of Resource Sha ing," *Proc. of the 1981 International Conference on Parallel Proces ing*, August 1981, pp. 162-163.

[LEI84] Dennis W. Leinbaugh, "Selector: High-Level Resource Schedulers, *IEEE Trans. on Software Engineering*, Vol. SE-10, No. 6, Novembe 1984, pp. 810-824.

[LIS81] B. H. Liskov, et al., *CLU Reference Manual (Lecture notes in Con puter Science)*, Vol. 114, Springer-Verlag, 1981.

[LIS82] B. H. Liskov, "On Linguistic Support for Distributed Programs, *IEEE trans. on Software Engineering*, Vol. SE-8, No. 3, May 198 pp. 203-210.

[MAG81] B. Maglaris, T. Lissack, and M. Austin, "End-to-End Dela analysis on Local Area Networks : An Office Building Scenario, *Proc. National Telecom Conf., 1978.*

[MAN84] R. Manner, "Hardware Task/Processor Scheduling in a Polyproce sor Environment," *IEEE Transactions on Computers*, Vol. C-3 No. 7, July 1984, pp. 626-636.

[MAR82a] M. A. Marson, and D. Roffinella, "Nonpersistent M-CSMA Proto cols for Multichannel Local Area Networks," *Proc. of the 7 Conference on Local Computer Networks*, Minnesota, Oct. 1982, p 100-109.

[MAR82b] M. A. Marson, and Gerla, "Markov Models for Multiple Bus Mu tiprocessor Systems," *IEEE Transactions on Computers*, Vol. C-3

No. 3, March 1982, pp. 230-248.

[MCC82] E.H. McCall, "Performance Results of the Simplex Algorithm for a Set of Real-Word Linear Programming Models," *Comm. of ACM*, Vol. 25, No. 3, March 1982, pp. 207-213.

[MET76] R.M. Metcalfe, and D.R. Boggs, "Ethernet : Distributed Packet Switching for Local Computer Networks," *CACM*, Vol.19, July 1976, pp.395-404.

[MIT81] K. Mittal, and A. Venetsanopoulos, "On the Dynamic Control of the Urn Scheme for Multiple Access Broadcast Communication Systems," *IEEE Trans. Communications*, Vol. COM-29, July 1981, pp. 962-970.

[MOK79] A.K. Mok, and S.A. Ward, "Distributed Broadcast Channel Access," *Computer Networks*, 1979, pp. 327-335.

[MOL82] M.L. Molle, "On the Capacity of Infinite Population Multiple Access Protocol," *IEEE Tran. on Information Theory*, Vol. IT-28, No.3, May 1982, pp. 396-401.

[MOS82a] J. Mosely, *"An Efficient Contention Resolution Algorithm for Multiple Access Channels,"* M.S. Thesis, Dept. of Electrical Engineering and Computer Sciences, MIT, Cambridge, May, 1982

[MOS82b] J. Mosely, and P. Humblet, "A Class of Efficient Contention Resolution Algorithms for Multiple Access Channels," *M.I.T., Report LIDS-P-1194*, March 1982.

[MOS85] J. Mosely, and P. Humblet, "A Class of Efficient Contention Resolution Algorithms for Multiple Access Channels," *IEEE Trans. on Comm.* Vol. C-35, Feb. 1985, pp. 145-157.

[NI81] L.M. Ni and K. Hwang, "Optimal Load Balancing Strategies for a Multiple processor System," *Proc. of 10th Int'l Conf. on Parallel Processing*, Aug. 1981, pp. 352-357.

[NI83] L. M. Ni, and X. Li, "Prioritizing Packet Transmission in Local Multiaccess Networks," *Proc. of Eighth Data Communications Symposium*, Cape Cod, MA, 1983.

[OUS80] J.K. Ousterhout, D.A. Scelza, and P.S. Sindhu, "Medusa: An Experiment in Distributed Operating System Structure," *CACM*, Vol. 23, No. 2, Feb. 1980, pp. 92-105.

[PAP65] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, N.Y. 1965.

[PAP82] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[PAT81] J. H. Patel, "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Trans. on Computers*, Oct. 1981, pp. 771-780.

[PIP81] N. Pippenger, "Bounds on the Performance of Protocols for a Multiple Access Broadcast Channel," *IEEE Trans. on Information Theory*, Vol. IT-27, No. 2, March 1981, pp. 145-151.

[PRE78] F.P. Preparata, "New Parallel-Sorting Schemes," *IEEE transactions on Computers*, Vol. C-27, Sept. 1975, pp. 669-673.

[REI79] E.M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms*, Prentice Hall, 1979.

[RIC74] D.M. Riche, and K. Thompson, "The Unix Time-Sharing System," *CACM*, Vol. 17, No. 7, July 1974, pp. 1278-1308.

[ROM81] R. Rom, and F.A. Tobagi, "Message-Based Priority Function i Local Multiaccess Communication Systems," *Computer Network* North-Holland Publishing Company, 1981, pp.273-286.

[SEG82] A. Segall, "Decentralized Maximum Flow Protocols," *Network* Vol. 12, 1982, pp213-230.

[SEL83] P. G. Selinger, "State-of-the-Art Issues in Distributed Databases, *IEEE trans. on Software Engineering*, Vol. SE-9, No. 3, May 198 pp. 218-219.

[SHA83] N. Shacham, "A Protocol for Preferred Access in Packet-Switchin Radio Networks," *IEEE Trans. Communications*, Vol. COM-31, N 2, Feb. 1983, pp. 253-264.

[SHI81] Y. Shiloach, and Uzi Vishkin, "Finding the maximum, Merging and Sorting in a Parallel Computation Model," *Journal of Algorithms, March 1981, pp. 88-102.*

[SHO82] J.F. Shock, et al., "Evolution of the Ethernet Local Computer Network," *IEEE, Computer,* Aug 1982, pp. 10-27.

[SIE81] H.J. Siegel and R.J. McMillen, "The Multistage Cube : A Versatile Interconnection Network," *IEEE Computer,* Vol. 14, No. 12, Dec. 1981, pp. 65-76.

[SIL83] A. Silberschatz, "Extending CSP to Allow Dynamic Resource Management," *IEEE trans. on Software Engineering,* Vol. SE-9, No. 4, July 1983, pp.527-530.

[SPE81] A.Z. Spector, "Performing Remote Operations Efficiently on a Local Computer Network," *CACM,* Vol. 25, No. 4, April 1982.

[STA84] W. Stallings, *Local Networks - An Introduction,* Macmillan Publishing Company, New York, 1984.

[STO77] H. S. Stone, "Multiprocessor Scheduling with the aid of Network Flow Algorithms," *IEEE Trans. Soft. Eng.* Vol. SE-5, No. 1, Jan. 1977, pp.85-93.

[THO73] R. H. Thomas, "A Resource Sharing executive for the ARPANET," *Proc. of National Computer Conference,* AFIPS press, 1973, pp. 155-163.

[THO77] C. D. Thompson, and H. T. Kung, "Sorting on A Mesh-Connected Parallel Computer," *CACM,* Vol. 20, No. 4, April 1977, pp. 263-271.

[THU72] K.J. Thurber, et al., "A Systematic Approach to the Design of Digital Bussing Structures," *Proc. of National Computer Conf.,* AFIPS press, Vol. 41, 1972, pp. 719-740.

[TAN81] A. S. Tanenbaum, *Computer Networks,* Prentice Hall, Inc., New Jersey, 1981.

[TOB82] F. A. Tobagi, "Carrier Sense Multiple Access with Message-Based Priority Functions," *IEEE Trans. Communications,* Vol. COM-30, No. 1, January 1982.

[TOD82] K. W. Todd, "Function Sharing in a Static Data Flow Machine, *Proc. of the 1982 International Conference on Parallel Processin* August 1982, pp. 137-139.

[TOW80] D. Towsley, "Queueing Network Models with State-Depender Routing," *JACM,* Vol. 27, No. 2, April 1980, pp. 323-337.

[TOW82] D. Towsley, and Venkatesh, G., "Window Random Access Protoco for Local Computer Networks," *IEEE Trans. Computers,* Vol. C-3 No. 8, August 1982, pp. 715-722

[TOW84a] D. Towsley and J.K. Wolf, "On Adaptive Tree Polling Algorithms *IEEE Trans. on Communications,* Vol. COM-32, No. 12, Decemb 1984, pp. 1294-1298.

[TRE82] P.C. Treleaven, D.R. Brownbridge and R.P. Hopkins, "Data-Driv and Demand-Driven Computer Architecture," *ACM, Computi Surveys,* Vol. 14, No. 1, March 1982.

[TSY79a] B.S. Tsybakov, and V.A. Mikhailov, "Free Synchronous Pack Access in a Broadcast Channel with Feedback," *Probl. Infor Trans.,* Vol. 14, 1979, pp. 259-280.

[TSY79b] B.S. Tsybakov, M.A. Berkovshi, N.N. Vedenskaja, V.A. Mikhailo and S.P. Fedorzov, " Methods of Random Multiple Access," *Fi International Symposium Inform. Theory,* July 7-9, 1979.

[WAN85] Y. T. Wang, and Robert J. T. Morris, "Load Sharing in Distribut Systems," *IEEE Transactions on Computers,* Vol. C-34, No. March 1985, pp. 204-207.

[WAH82] B.W. Wah and A. Hicks, "Distributed Scheduling of Resources Interconnection Networks," *Proc. National Computer Conferen* AFIPS Press, 1982, pp. 697-709.

[WAH83a] B.W. Wah, "A Comparative Study of Distributed Resource Shari on Multiprocessors," *Proc. of 10th Annual International Symposi on Computer Architecture,* 1983, pp. 300-308c.

[WAH83b] B.W. Wah and J.Y. Juang, "Load Balancing on Local Multiaccess Networks," *Proc. of 8th Conference on Local Computer Networks*, Oct. 1983, pp. 56-66.

[WAH84a] B. W. Wah, and K. L. Chen, "A Partioning Approach to the Design of Selection Networks," *IEEE, Transactions on Computers*, Vol. C-33, No. 3, March 1984, pp. 261-268.

[WAH84b] B.W. Wah, "A Comparative Study of Distributed Resource Sharing Interconnection Networks on Multiprocessors," *IEEE Trans. on Computers*, Vol. C-33, No. 8, Aug. 1984, pp. 700-711.

[WAH84c] B. W. Wah, and J. Y. Juang "An Efficient Contention-Resolution Protocol for Local Multiaccess Networks," United States Patent Pending, 1984.

[WAH85] B. W. Wah, and Y. N. Lien, "Design of Distributed Databases on Local Computer Systems with Multiaccess Networks," *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 7, July 1985, pp. 606-619.

[WAL83] B. Walker, et al, "The LOCUS Distributed Operating System," *Proc. of Nineth ACM symp. on Operating System Principles*, 1983, pp.49-70.

[WAN85] Y.T. Wang, and J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. on Computers*, Vol. C-34, No. 3, March 1985, pp. 204-217.

[YAO80] A. C. C. Yao, "Bounds on Selection Networks," *SIAM J. Comput.* Vol. 9, No. 3, pp. 566-582, August 1080.

APPENDIX A

Load Balancing on Systems Connected by A CSMA/CD Network

A.1 Load Balancing

Load balancing is a scheme that engages the communication facilities t
support remote job execution in a user-transparent manner in order t
improve utilization of resources. Depending on the workload of processors, th
network operating system may distribute newly arrived jobs to a remote pro
cessor, or may schedule them for local execution. The concept of load balanc
ing has been implemented in the Engineering Computer Network at Purdu
University [HWA81]. Operational experiences with the system reveal that loa
balancing has contributed significantly to the improved resource utilization an
reduced response time.

Theoretical studies on load balancing have been done with respect to cen
tralized control [CHO79, TOW80, NI81]. Newly arrived jobs are routed t
different processors by a scheduling policy. Two classes of scheduling strategi
have been investigated, namely, *probabilistic* and *deterministic*. In a probabili
tic routing strategy, an arriving job is dispatched to the processors according t
a set of branching probabilities. These probabilities are associated with t
statistics gathered about the processing speeds and job arrival history. On t

other hand, a deterministic strategy is a state-dependent scheduling strategy. It routes jobs to the processors according to the current workload status. This is similar to a probabilistic strategy except that the branching probabilities are updated dynamically whenever the system state changes.

Previous studies on both strategies have shown that a multiprocessor system with load balancing outperforms one without it, and its performance is nearly as good as that of a single fast processor. However, three problems occur when these strategies are applied to a system connected by a relatively slow network. First, the success of the above strategies lies in the fact that the information on system load are readily available without any delay. When the network is slow, significant delays may be incurred in distributing the status information. The routing of jobs based on inaccurate information may have an adverse effect on performance. Second, the distribution of status information may have a large overhead on the network traffic. Special considerations may have to be made to reduce this overhead based on the network characteristics. Third, migration of tasks tends to increase the network load and impede message traffic. Tradeoffs must be made to balance the delay of jobs and messages.

Task allocation in a distributed computer network is also a load-balancing scheme [STO77, CHU80, CHO82, WAN85]. It deals with finite number of mutually dependent processes which carry out a task cooperatively. Interprocesse communication is crucial to the perforamnce of such distributed processes. The problem of task allocation is to allocate these processes to available processors properly so that the execution cost is minimized. This class of load balancing is out of the scope of this thesis.

A.2 MAX-MIN Load Balancing Strategy

Due to the limitation of CSMA/CD networks, only one job can be ser over the network at any time. According to the optimal scheduling algorith developed in Chapter 3, jobs from the processor with the heaviest load shou migrate to the processor with the lightest load. This scheduling algorithm called the MAX-MIN load balancing strategy in this appendix. To support tl MAX-MIN load balancing strategy, the following tasks are involved:

(a) to decide when load balancing is needed;

(b) to identify the processors with the maximum and minimum load;

(c) to send a job from the processor with the maximum load to the process with the minimum load;

(d) to send the results of execution back to the originating site.

Migrating jobs and returning results can be handled in a similar way regular message transfers. However, the decision to perform load balancing not depends on the network traffic. If the total round-trip delay of send jobs over the network and the execution time on the remote processor is lon than the local execution time, then load balancing is not beneficial. Monit on network traffic must be kept at each processor. When it is decided t load balancing is necessary based on the previous status information collect the workload status of all processors must be determined to identify the sou and sink of the job to be balanced. A simple way is to scan the processors round-robin fashion. This is time-consuming because it depends on the num of processors in the network. We describe in the next section an efficient tocol to resolve this problem.

A.3 A CSMA/CD Protocol for Supporting MAX-MIN Load Balancing Strategy

There are four types of tasks to be serviced by the protocol for supporting MAX-MIN load balancing. These include regular message transfer, result return, job migration, and identification of the processors with the heaviest and the lightest loads (MAX-MIN identification). They are listed here in descending order of priorities. Since the network was originally designed for message transfers, we assign the highest priority to messages. Load balancing can be done by using the spare channel capacity. The return of results from a remote processor is important because its delay contributes to the response time of jobs. It is also natural that previous load balancing process should have higher priority than that of a newly invoked one. Thus, the MAX-MIN identification is assigned the lowest priority.

The identification of the processor for using the channel consists of two phases. First, the group of processors with the highest priority must be determined. After these processors are known, a contention phase follows, which identify a unique processor to transmit. These phases must be carried out for every packet sent in the system. A summary of the protocol is shown in Figure A.1.

The effect of our priority assignment favors the transfer of messages. However, it would reduce the possibility of load balancing in the system. Suppose that the maximum (resp. minimum) response time in the system is r_{max} (resp. r_{min}), and that the transmission delay of sending a job over the network is s_j. Let the channel load for messages be c_m. Since messages have higher

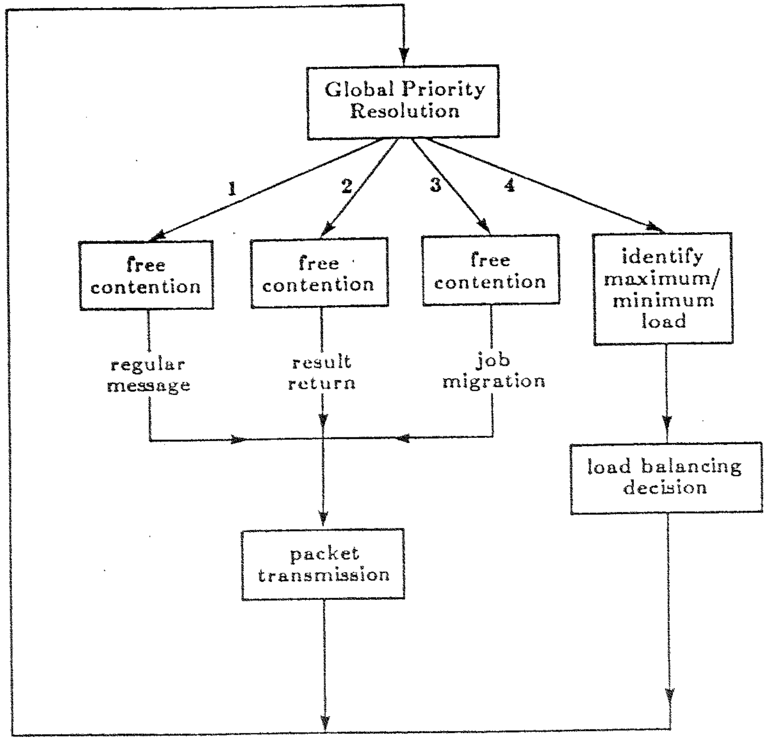


Figure A.1 A protocol to support MAX-MIN load balancing in CSMA/CD n works

priority, the average total delay of sending a job is $s_j/(1-c_m)$. Assuming that the return of results takes the same amount of time, load balancing will be effective if

$$r_{max} > r_{min} + \frac{2\,s_j}{1 - c_m}$$

or

$$c_m < 1 - \frac{2\,s_j}{r_{max} - r_{min}} \qquad \text{(A.1)}$$

That is, although the channel is idle, load balancing cannot be carried out because the channel delay is too high. A way to improve the channel delay for load balancing is to reduce the priority of messages. This reduces the average delay of job and result transfer at the expense of increased message delay.

The tasks required in the MAX-MIN load balancing protocol can be considered as the problem of finding the minimum of a set of physically distributed random numbers. (Finding the maximum can be transformed into finding the minimum using an inverse function.)

(a) Contention resolution for packet transmission: When multiple processors wish to transmit, each processor generates a random number in the interval (0,1]. The processor assigned the channel is the one with the minimum number.

(b) Identification of processor with the maximum (resp. minimum) load: The load on a processor is characterized by the response time of executing a job on that processor. The response times of all processors are distributed in the interval $[0,\infty)$. To find the processor with the maximum (resp. minimum) load is equivalent to finding the maximum (resp. minimum) of

a set of random numbers in the interval $[0,\infty)$. To apply the window con trol schemes developed in Chapter 3, the distribution of response times ha to be known. It can be estimated emperically. Due to the effect of loa balancing, the response time of processors are not independent. Howeve they are assumed to be independent here for mathematical tractability.

(c) Identification of the processors with the maximum priority: The priority a processor is an integer from the set $\{0, 1, \ldots, p_{max}\}$. The identificatio problem requires the maximum priority value of all processors in th interval to be found.

These problems can be generalized as follows. Given a set of independe random numbers y_i, $i = 1, 2, \ldots, n$, with distribution $F(y)$ such that y_1 $y_2 \leq \cdots \leq y_n$, the problem is to identify the processor(s) with value Transformation may have to be performed on the original set of rando numbers in order to result in the required distribution. This problem can solved efficiently using the distributed minimum-search algorithm developed Chapter 3. It should be noted that in many cases, only an estimate on n c be obtained.

A.4 Performance of MAX-MIN Load Balancing

A simulator has been developed to evaluate the performance of l balancing as supported by the window protocol. The simulator was writter ASPOL, and assumed that job arrivals are Poisson at each processor and t the service time is exponentially distributed. The normalized mean response time with respect to different job traffic with no message traffic is plotted in Figure A.2. The results are compared against the case without load balancing, that is, the case of multiple independent M/M/1 queues. We find that the average response time is always improved when load balancing is applied.

Load balancing is less effective when the job load is low, since almost no queue is developed at each processor. In Figure A.3, we show the job response time with respect to the ratio of job service time and transmission delay. The percentage of jobs being migrated is also shown. As we can interpret from this figure, a system with fast processors coupled with a slow communication channel (r is small) will not benefit from load balancing. As the speed of the communication channel increases, more jobs are migrated and the response time is improved. However, when the channel speed exceeds a threshold, load balancing can provide no further improvement. In Figure A.4, the effects on load balancing due to message traffic is shown. As the message traffic becomes higher, the effective channel capacity is less, and load balancing is not carried out as often.

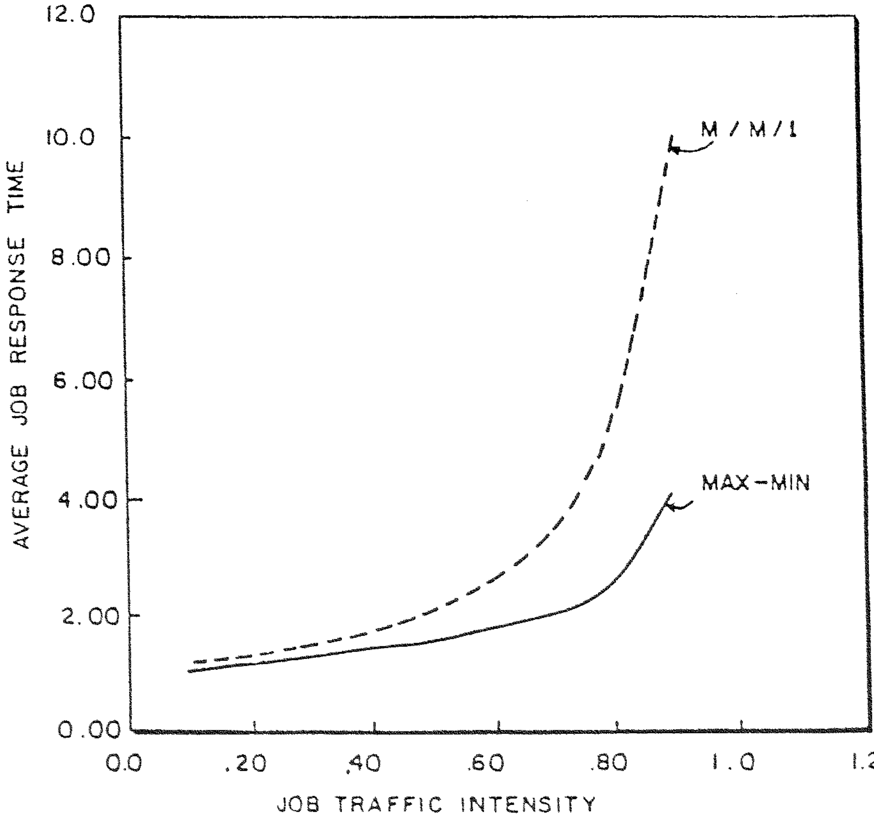


Figure A.2 A reduction in job response time under load balancing with no message traffic

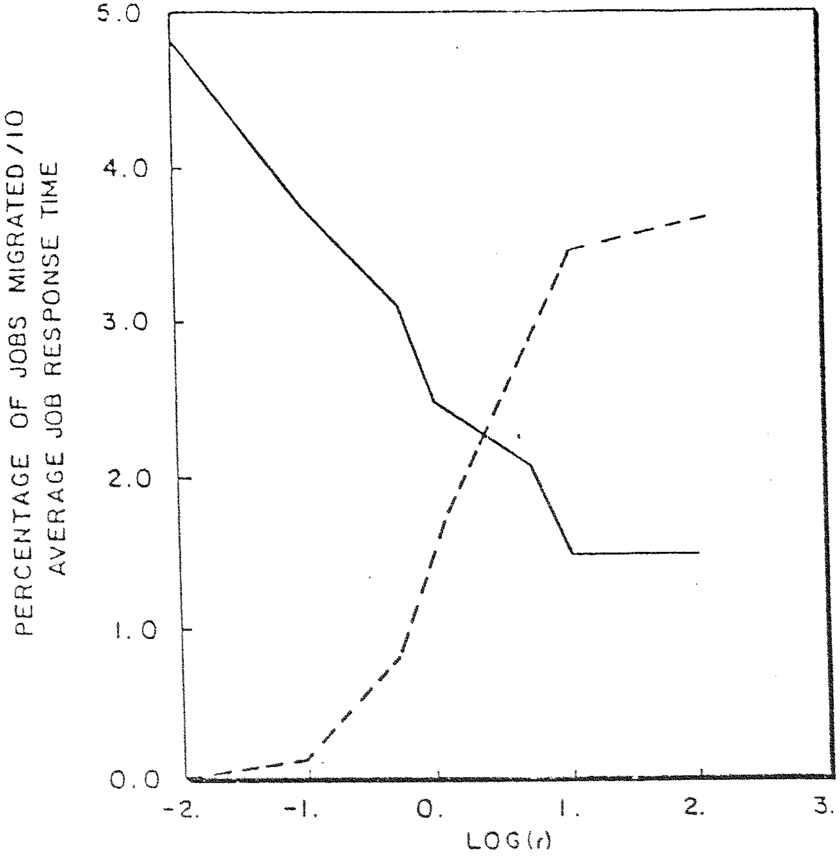


Figure A.3 Effects of the ratio of service to transmission delays

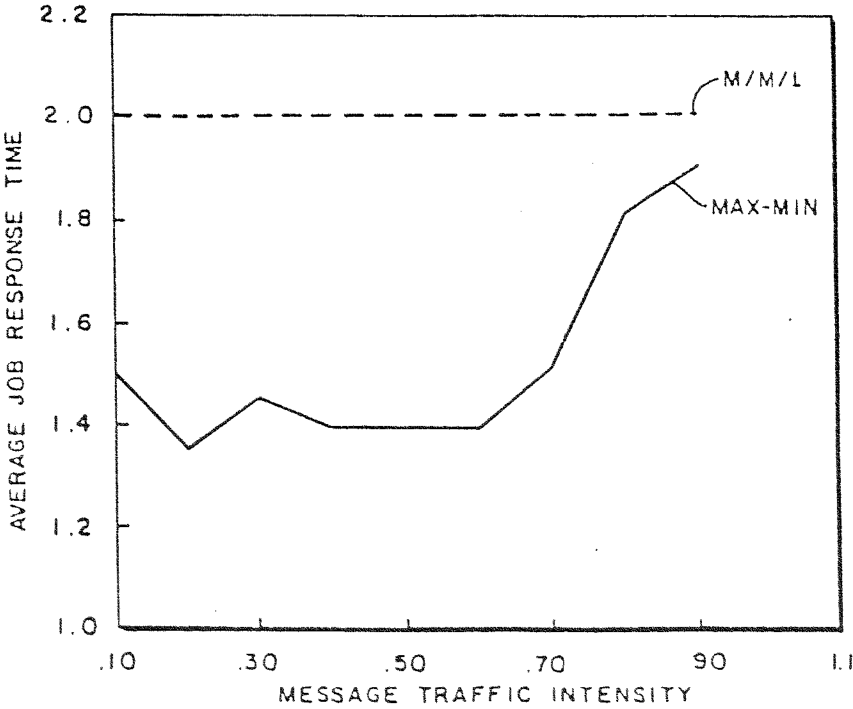


Figure A.4 Effects on load balancing due to message traffic

APPENDIX B

Unification of Adaptive CSMA/CD protocols

B.1 Introduction

The contention-resolution problem can be regarded as a problem of determining a sequence of enabled set of stations until the last enabled set contains only one active station. The problem has been studied for over a decade, and many protocols were proposed. These protocols can broadly be classified into nonadaptive and adaptive according to their contention-resolution algorithms. A nonadaptive CSMA protocol selects an enabled set regardless of the channel load. Nonpersistent and p-persistent protocols are examples of this class [KLE75]. On the other hand, the knowledge of the channel load is used to decide the enabled set in each contention slot in an adaptive CSMA protocol. This knowledge may be estimated explicitly by a dedicated channel monitor as proposed by Kleinrock and Yemini [KLE78], or may be acquired implicitly by the contention experience of an individual station as in the Binary Exponential Backoff scheme of Ethernet [MET76]. It has been shown that an infinite population CSMA network with nonadaptive protocols is inherently instable [FAY77]. The effective throughput of the channel will tend to zero due to the possible endless collisions. To maintain a stable channel, an adaptive protocol is necessary.

The adaptive contention-resolution problem with ternary feedback is isomorphic to a class of sequential decision problems [HLU81]. Although many efforts have been made in searching for an optimal solution to such problems, and Markov decision processes have been applied to maximize the channel throughput for some specific protocols, they are computationally inefficient due to the large state space [HLU81,MOS82]. Alternatively, various bounds on the maximum achievable channel throughput have been established [PIP81, TSY79a, TSY79b, CRU82, MOL82, HAJ82]. A closer look at the problem reveals that the large degrees of freedom in determining the enabled set is the major source of complexity. In this section, we present the class of protocols with restrictions in determining the enabled sets. The class of protocols are isomorphic to the window protocol presented in Chapter 3. Many existing protocols will be shown to belong to the class of window protocols. Notably are the Adaptive-Tree-Walk Protocol [CAP79a,CAP79b], the Urn Protocol [KLE78, MITT81], the Arrival-Time-Window Protocol [GAL78, MOS82, TOW82, KUR83, KUR84], and the Virtual Window Protocol [WAH83]. In Section B.2 a distributed window search procedure is described, the various adaptive CSMA protocols are briefly reviewed, and their mappings to the window protocol are identified. In particular, the underlying distributions of the contention parameters are derived, and the schemes for estimating the distributions and channel load are proposed.

B.2 Characteristics of The Window Protocols

The protocols we investigate have the following three restrictions : (1) a linear ordering relation holds for the station identifiers (station-ID in short), (2) the active station with the minimum station-ID is given channel access, and (3) the enabled set is convex in the station-ID domain. Since enabled stations constitute a convex set in the station-ID domain and is analogous to a window in this domain, so this class of protocols is named *window protocols*. Notice that this class of protocols is larger than that of Hluckyj's classification [HLU81] which consists systems of finite population and discrete station-IDs, although the same name is also used. Since a station-ID is assigned dynamically, it will be called the contention parameter in the sequel.

In the class of window protocols, the contention-resolution problem is equivalent to the problem of finding the extremum among a set of distributed random numbers.

B.3 The Family Of Window Protocols

In this section, several adaptive CSMA/CD protocols are shown to be members of the family of window protocols. The contention parameters that characterize these protocols will be discussed here. The distributions governing the generation of the contention parameters are derived because they are crucial to the optimization of window control.

B.3.1 Adaptive-Tree-Walk Protocol

This protocol is basically a preorder tree traversal algorithm [CAP79]. A tree is organized in such a way that each leaf is associated with a station in th network. The search begins at the root, and all ready stations can transmit i the first contention slot. If a collision occurs, the search continues recursivel from the leftmost subtree to the rightmost subtree in the following contentio slots. The search is successful when a single active station is contained in subtree, which is allowed to transmit its packet without collision. After th packet is transmitted, the next contention slot is reserved for the next subtre in the preorder search.

To have better performance, the size of the enabled subtrees should ada to the channel load. A small subtree is enabled when the channel load heavy, and a larger one is enabled if the channel load is low. One way adapt to different traffic conditions is to have different degree of internal nod under different traffic conditions. Capenakis proposed a recursive scheme construct a tree that maximizes the channel efficiency [CAP79a,CAP79b]. T scheme assumes that the network operates in steady state known arrivals th are Poisson distributed. It also requires that channel loads are estimat separatedly, and that the search tree has to be reconstructed when the chan load changes.

This protocol can be viewed as a moving window protocol. The leaves the tree are labeled from 1 to N. The origin of the window is rotated dyna cally across the leaves. A window is equivalent to an enabled subtree. T objective is to isolate an active station with the minimum distance from origin by the window-search procedure.

From the model of the window protocol, the leaves of the tree is labeled from 1 to N starting from the origin. Station i generates a contention parameter equal to the distance from station 1 (the origin) if it is active, and generates a very large number, say N+1, if it is idle. The probability density function from which a station generates its contention parameter is

$$f_i(k) = \begin{cases} \Pr\{\text{i'th station is active}\} & k = i \\ \Pr\{\text{i'th station is idle}\} & k = N+1 \\ 0 & \text{otherwise} \end{cases} \qquad (B.1)$$

Let p_i be the probability that the i'th station becomes active in one unit of time if it was previously idle, and t_i be the elapsed time since the station was last enabled to transmit. In the following derivation, we assume that there is only one buffer at each station and that the probability for an idle station to become active in one unit time is constant for all stations i.e. $p_i = p$ for all i. Then,

$$\Pr\{\text{i'th station is active} \mid p, t_i\} = 1 - (1-p)^{t_i} \qquad (B.2)$$

On the other hand, p may be expressed as a function of the total number of stations in the network (N) and the number of stations that have transmitted after the window has circumscribed around all the stations once (n). Assuming that it takes θ_n units of time to transmit n packets, the next station allowed to transmit must have been waiting for θ_n units of time, and the probability that the station is active is $(1-(1-p)^{\theta_n})$. Denote the status of the i'th station by a random number X_i such that the value of X_i is one if the station is active and zero otherwise. Since Station i has been waiting for θ_n units of time,

$$\Pr(X_i=1) = 1-(1-p)^{\theta_n} \qquad (B.3$$

From probability theory and Eq. B.3, n may be expressed as follows.

$$n = \sum_{i=1}^{N} X_i$$

$$= N\,(1-(1-p)^{\theta_n}) \qquad (B.4$$

If the contention time is much smaller than a packet-transmission time, then $\theta_n \simeq n$. So we have the following relation :

$$N\,(1-(1-p)^{n}) = n \qquad (B.5$$

Solving Eq. B.5 yields:

$$p = 1 - \left(1 - \frac{n}{N}\right)^{\frac{1}{n}} \qquad (B.6$$

By replacing p in Eq. B.2 by the RHS of Eq. B.6, we obtain the distributio that governs the generation of contention parameter by the i'th station as fol lows:

$$F_i(k) = \begin{cases} 0 & k < i \\ 1 - \left[1 - \frac{n}{N}\right]^{\frac{t_i}{n}} & i \leq k \leq N \\ 1 & k > N \end{cases} \qquad (B.7$$

In Eq. B.7, t_i and n are unknown variables. To estimate these variables an N-bit register is introduced at every station to record the status of all th stations during the last circumscription by the window. All the registers ar updated synchronously as follows. After each contention is resolved, th station, say i, that wins the contention is identified as the only active station among those enabled. The i'th status bit of the register is set to 1, and the bits associated with other enabled stations are set to 0. Consequently, the channel load n is the total number of bits with value 1, and the elapsed time of a station since its last transmission may also be obtained by counting the number of bits being set to one between the station and the origin of the window.

In case that the register is too expensive to maintain, the channel load, n, can still be obtained by counting the average number of stations that transmit during a circumscription by the window. However, t_i cannot be found exactly, but can be estimated from n. With the assumption of uniform packet arrivals, the probability that a packet was transmitted in the last circumscription by the window with distance greater than i from the current origin of the window is (N−i)/N. The number of packets transmitted in the last circumscription follows an incomplete binomial distribution. To compute the expected value of the elapsed time t_i, it is necessary to know the minimum and maximum number of stations that could have transmitted when the window has moved from Station i to the current origin. The maximum cannot exceed either n or N−i, and the minimum cannot be less than either 0 or n−i. Thus, the probability that k out of n packets being transmitted after Station i is

$$\Pr(t_i=k) = \frac{\binom{n}{k}\left(\frac{N-i}{N}\right)^k\left(1-\frac{N-i}{N}\right)^{n-k}}{\sum_{j=\max(0,n-i)}^{\min(n,N-i)}\binom{n}{j}\left(\frac{N-i}{N}\right)^j\left(1-\frac{N-i}{N}\right)^{n-j}} \qquad \text{(B.8)}$$

Substituting Eq's B.2, B.6 and B.8 into Eq. B.1, the distribution function for

Station i, $1 \leq i \leq N$, to generate its contention parameter is

$$F_i(k) = \begin{cases} 0 & k < i \\ 1 - \sum_{j=\max(0,n-i)}^{\min(n,N-i)}\left(1-\frac{N-i}{N}\right)^{j/n}\Pr(t_i=j) & i \leq k \leq N \\ 1 & k > N \end{cases} \qquad \text{(B.9}$$

B.3.2 Urn Protocol

The Urn Protocol of Kleinrock and Yemini [KLE78] is a window protoc in the spatial domain. It assumes that n out of N stations are active, and th Urn model in probability theory is applied to find an appropriate number enabled stations such that the probability of exactly one active station i enabled set is maximized. Binary-divide is used if there are more than one st tion in the set. In this approach, active stations are assumed to be uniform distributed among all stations. Kleinrock and Yemini proposed a method maintain such an uniformity, in which the enabled set is determined by sy chronized pseudo random number generators. Equivalently, the stations a labeled according to a pseudo random sequence. Synchronization in the ne work environments is nontrivial however.

Alternatively, a rotating window approach is proposed in which the initi window size is also determined by the Urn model. This approach is similar the Adaptive-Tree-Walk Protocol in that the stations are enabled sequentiall However, due to the round-robin service discipline, those stations closer to t origin of the window have a longer elapsed time since last enabled, and thus higher probability of becoming active. In this model, the active stations are no longer uniformly distributed over the station space. The Urn protocol has also to be supported by a subchannel to estimate channel load n. However, the channel load can be estimated easily from the window sizes of the protocol. To obtain a sequence of optimal windows for contention resolutions, the scheme proposed for the Adaptive-Tree-Walk may be used.

B.3.3 Priority-CSMA Protocol

Accesses with priority are important in scheduling shared resources. In particular, the communication channel is a valuable resource in the network and should be accessed with priorities. Several CSMA protocols for handling priority messages have been suggested in recent years [TOB82, GOL83, NI83, SHA83]. They may be classified as linear protocols and logarithmic protocols. Each station is assigned the highest priority of the local messages. In a linear protocol, a slot is reserved for each priority level during the resolution of priorities. An active station contends during the slot reserved for the local priority level. The process stops when the highest priority level is determined. This scheme is good when high-priority messages are predominantly sent. A logarithmic protocol determines the highest priority level in $O(\log_2 P)$ steps by a binary-divide scheme, where P is the maximum number of priority levels [NI83]. This assumes that the highest priority level is equally likely to be any one of the P priority levels. Neither of the schemes is able to adapt to the various traffic patterns.

The global priority resolution problem can be solved by a window protoc such that the highest priority level at each station is used as the contenti parameter [WAH83]. To determine the highest priority level being present the network, a subset of the high-priority levels is chosen, and stations wi local priority belonging to this subset are allowed to contend. Since there m be more than one station in each priority level, a collision may happen due transmissions from either stations of the same priority level or stations different levels. The global priority level is resolved if there is only one priori level in the enabled set in spite of collisions.

To apply our window-optimization method, it is necessary to identify t distribution of the presence of each priority level. Let λ_i be the arrival rate the i'th priority level, and t_i be the arrival time of the most recent packet the i'th level that has been transmitted. Assuming a Poisson process for t packet arrivals, then the probability that there is at least one station in the i' priority level is

$$p_i = 1 - \int_{T-t_i}^{\infty} \lambda\, e^{-\lambda t} dt$$

$$= e^{-\lambda(T-t_i)} \qquad \text{(B.1}$$

where T is the current time. Hence, the distribution of priority level i is

$$F_i(k) = \begin{cases} 0 & k < i \\ e^{-\lambda_i(T-t_i)} & i \leq k \leq L \\ 1 & k > L \end{cases} \qquad \text{(B.1}$$

where L is the maximum priority level. The arrival time of a packet may be acquired by piggybacking information on the packet. The packet arrival rate may be estimated by observing the packet arrival times.

The distribution of the contention parameters in the priority-CSMA protocol is of the same structure as that of the Adaptive-Tree-Walk protocol. Hence, these protocols will have the same performance.

B.3.4 Arrival-Time-Window Protocol

Gallagher proposed a window protocol on the time axis in which all stations have a common window of length u in the past [GAL79]. Stations with packets arriving within the window of time are allowed to contend. If there is no transmission or a successful transmission, the window is advanced u units of time; otherwise, the window is reduced to a fraction, f, of its original size, and the process is repeated until a packet is transmitted successfully. The parameters u and f are chosen to optimize the performance. Binary-divide is used in Gallagher's protocol. Subsequently, Mosley applied the non-discounted Markov decision process to refine the protocol and achieved the highest channel utilization so far [MOS82a, MOS85]. Towsley relaxed the constraint of collision detection and assumed that the exact number of stations involved in a collision is known. A recursive procedure is used to maximize the channel efficiency, but little improvement has been obtained [TOW82, KUR83, KUR84]. Although these protocols have high efficiency theoretically, there are several drawbacks. First, the steady-state assumption is made in the optimization and

the instantaneous load conditions are not reflected in the problem. Second, t optimization techniques are computationally inefficient. Lastly, the chann load (packet-arrival-rate) estimation method is not included in the protocol.

The distributions that govern the generations of contention parameters a derived in the following. Suppose that the window begins at time O, that t current time is T, and that there are n contending stations. Since the proto searches for the earliest packet-arrival time in this window, each station c use the earliest packet-arrival time in the interval (O,T) as the contenti parameter. Assuming the arrival process at each station to be Poisson, the d tribution of the earliest arrival time conditioned on the current time T and t origin of window O is:

$$F_i(t \mid O<t\leq T) = \begin{cases} 0 & t\leq O \\ \dfrac{1-e^{-\lambda_i(t-O)}}{1-e^{-\lambda_i(T-O)}} & O<t\leq T \\ 1 & t>T \end{cases} \quad i=1,\ldots,N \qquad \text{(B.}$$

where λ_i is the packet arrival rate at station i. Notice that if $\lambda_i \neq \lambda_j$, t $F_i(t) \neq F_j(t)$. The inference method discussed in Chapter 3 may be used estimate the number of contending stations.

B.3.5 Virtual-Window Protocol

A new Virtual-Window Protocol has been proposed to resolve content of messages [WAH83b, WAH84c]. Each of the n active stations generat random number from the uniform distribution U(0,1) as its conten parameter. That is,

$$F_i(y) = \begin{cases} 0 & y \leq 0 \\ y & 0 < y \leq 1 \quad i = 1, \ldots, N \\ 1 & y > 1 \end{cases} \tag{B.13}$$

A contention parameter is only a dummy argument in this protocol, and no physical meaning is attributed. However, any application oriented parameters may be mapped into this virtual domain using a one-to-one mapping of distribution functions. The mapping is illustrated below by the Arrival-Time window protocol.

The contention parameter x_i of the Arrival-Time Window Protocol is generated by an incomplete exponential distribution. A random variable generated by such a distribution, $F_i(\cdot)$, can be transformed into another random variable that is uniformly distributed over (0,1) by replacing x_i with [PAP65]:

$$x_i' = F_i(x_i) \tag{B.14}$$

Since this transformation is a one-to-one mapping, and if the distributions are identical for all stations, then the optimization performed on the transformed contention parameters can be shown to be equivalent to the original optimization.

On the other hand, if the distributions are non-identical, some properties are lost after the transformation. For example, the original order of packet arrivals may not be preserved after the transformation. A packet arriving earlier at a station with a light traffic than a packet arriving at a station with a heavy traffic may be transformed into a larger contention parameter than that of the station with the heavy traffic. Due to this phenomenon, the First-

Come-First-Serve discipline proposed in [GAL79, KUR83, KUR84] cannot applied on the transformed contention parameters.

Load-estimation methods for this protocol were described in Chapter 3.

B.4 Performance of the Window Protocols

The problem of optimizing window control was formulated recursively a dynamic programming problem in Chapter 3 of this thesis. With the sa notation, the formulation is repeated here for convenience.

$$n(a,b) = \min_{a<w<b} \left\{ \begin{array}{c} 1 + 0 \cdot g(w,a,b) \\ + n(a,w) \cdot \ell(w,a,b) + n(w,b) \cdot r(w,a,b) \end{array} \right\} \qquad \text{(B.}$$

Let $F_i(x)$ be the distribution function for generating x_i, $1 \leq i \leq N$, and M the number of stations that are contending (M=N for the Tree-walk or U Protocols, M=n for other protocols), then event A has probability:

$$\Pr(A) = \frac{\prod_{i=1}^{M}[1-F_i(a)] - \sum_{i=1}^{M}\left\{[F_i(b) - F_i(a)] \prod_{\substack{j=1 \\ j \neq i}}^{M}[1-F_j(b)]\right\} - \prod_{i=1}^{M}[1-F_i(b)]}{\prod_{i=1}^{M}(1-F_i(a))} \qquad \text{(B.}$$

The first and last terms indicate the probabilities that all x_i's are greater t a and b, respectively. The second term is the probability that exactly one the x_i's is in the interval (a,b]. Similarly,

$$g(w,a,b) = \frac{1}{\Pr(A)\prod_{i=1}^{M}(1-F_i(a))} \sum_{i=1}^{M}[F_i(w)-F_i(a)] \qquad (B.17)$$

$$* \left\{ \prod_{\substack{j=1\\j\neq i}}^{M}[1-F_j(w)] - \prod_{\substack{j=1\\j\neq i}}^{M}[1-F_j(b)] \right\}$$

$$r(w,a,b) = \frac{1}{\Pr(A)\prod_{i=1}^{M}(1-F_i(a))} \left\{ \prod_{i=1}^{M}[1-F_i(w)] \right.$$

$$\left. - \sum_{i=1}^{M}\left[[F_i(b)-F_i(w)] \prod_{\substack{j=1\\j\neq i}}^{M}[1-F_j(b)] \right] - \prod_{i=1}^{M}[1-F_i(b)] \right\} \qquad (B.18)$$

The probabilities g(w,a,b), ℓ(w,a,b), and r(w,a,b) depend on the distributions of the contention parameters. The distributions obtained in Section B.3 were used to evaluate the associated protocols under the dynamic-programming window control. The values of n(a,b) are computed and compared with simulation results.

B.4.1 Virtual-Window and Arrival-Time Window Protocols

Since the distributions of the contention parameters are independent and uniformly distributed over (0,1), Eq's B.17 and B.18 can be reduced to simpler forms:

$$g(w,a,b) = \frac{(w-a)\,[(1-w)^{n-1}-(1-b)^{n-1}]}{(1-a)^n-(1-b)^n-n(b-a)(1-b)^{n-1}} \qquad (B.1$$

$$r(w,a,b) = \frac{(1-w)^n-(1-b)^n-n(b-w)(1-b)^{n-1}}{(1-a)^n-(1-b)^n-n(b-a)(1-b)^{n-1}} \qquad (B.$$

Numerical evaluations and simulations based on the above distribution ha been discussed in Chapter 3.

B.4.2 Tree-Walk and Urn Protocols

The discrete distributions for both the Tree-Walk and Urn Protocols non-identical and time-varying. Assuming that the origin of the window i Station 1, there are two properties that can be used to reduce the complex of Eq's B.17 to B.18: (i) $F_i(k)=0$ for $k<i$; and (ii) $F_i(a)=F_i(b)$ for $i \le a,b$ From these, we obtain:

$$\Pr(A) = \prod_{i=1}^{a}[1-F_i(a)] - \sum_{i=a}^{b}F_i(b)\prod_{\substack{j=1\\j\neq i}}^{b}[1-F_j(b)] - \prod_{i=1}^{b}[1-F_i(b)] \qquad (I$$

$$g(w,a,b) = \frac{1}{\Pr(A)}\sum_{i=a}^{w}F_i(w) * \left\{ \prod_{\substack{j=1\\j\neq i}}^{w}[1-F_j(w)] - \prod_{\substack{j=1\\j\neq i}}^{b}[1-F_j(b)] \right\} \qquad ($$

$$r(w,a,b) = \frac{1}{\Pr(A)}\left\{ \prod_{i=1}^{w}[1-F_i(w)] - \sum_{i=w}^{b}\left[F_i(b)\prod_{\substack{j=1\\j\neq i}}^{b}[1-F_j(b)] \right] - \prod_{i=1}^{b}[1-F_i(b)] \right\} \qquad ($$

Among the protocols we have considered, the Tree-walk and Urn Pro have the highest certainty about the values of the contention parameters. The performance is, therefore, expected to be the best. The average height of the dynamic programming tree has been evaluated with respect to the different channel loads. The approximate distribution of Eq. B.7 is used in the numerical evaluations. The results plotted in Figure B.1 verify that perfect scheduling is achieved when the load is heavy. It should be noted that the performance degrades as the total number of stations increases. When N→∞, the protocol behaves like one with a continuous distribution. The performance of the system was also simulated. The window-control and load-estimation schemes described in Section B.3.1 are implemented directly in the simulator. The distribution of contending parameters were estimated at each station using Eq. B.8. The average number of contending station estimated is called the *load average* in this simulator. In driving the simulation, a station generates a packet in a packet-transmission time with probability p. The simulation results are summaried in Tables B.1 and B.2. The results demonstrate that the proposed scheme is very efficient.

Since the distribution of the global priority-resolution protocol has the same structure as that of the Tree-Walk and Urn protocols, the performance evaluations discussed in Section B.4.3 also apply to the global priority-resolution protocol.

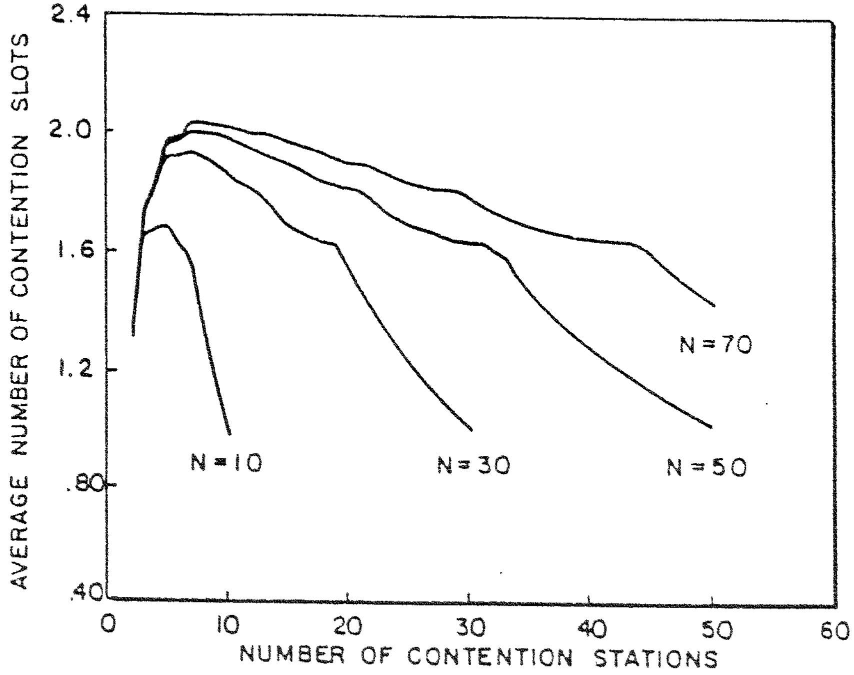


Figure B.1 Performance of the TREE-URN protocol with dynamic-programming window control

Table B.1 Simulation results of the TREE-URN protocol with dynamic-programming window control (number of stations = 10)

TREE-URN with Dynamic Programming Window Control (No. of Stations = 10)		
p	Load Ave.	# of Contentions
0.02	1.27	2.07
0.04	1.47	1.68
0.06	1.93	1.87
0.08	2.64	1.84
0.10	3.30	1.54
0.12	3.77	1.60
0.14	5.17	1.48
0.16	6.26	1.37
0.18	7.05	1.27
0.20	7.90	1.23
0.22	8.50	1.16
0.24	8.91	1.12
0.26	9.19	1.10
0.28	9.32	1.08
0.30	9.54	1.05
0.32	9.74	1.03
0.34	9.83	1.02
0.36	9.85	1.02
0.38	9.83	1.02
0.40	9.93	1.01

Table B.2 Simulation results of the TREE-URN protocol with dynamic-programming window control (number of of stations = 20).

TREE-URN with Dynamic Programming Window Control (No. of Stations = 20)		
p	Load Ave.	# of Contentions
0.01	1.22	2.21
0.02	1.50	2.38
0.03	2.39	2.09
0.04	3.21	1.59
0.05	3.89	1.82
0.06	6.70	1.59
0.07	7.72	1.47
0.08	10.22	1.55
0.09	12.64	1.37
0.10	14.57	1.28
0.11	16.10	1.21
0.12	16.96	1.18
0.13	17.50	1.14
0.14	18.10	1.10
0.15	18.76	1.07
0.16	19.09	1.04
0.17	19.16	1.01
0.18	19.37	1.03
0.19	19.51	1.02
0.20	19.54	1.02

B.5 Concluding Remarks

In this appendix, we have described the window-search scheme that determines the minimum among a set of distributed random numbers, and unifies a class of adaptive CSMA protocols. The unification allows the optimization of various protocols to be done by a unique method. Dynamic-programming formulation to minimize the expected total number of contention slots was studied and verified by simulations. The formulation was based on information on channel load and distributions of contention parameters. In practice, the channel load cannot be obtained directly and has to be estimated from the window size, the first-order statistic of the contention parameters, and the distributions of contention parameters.

VITA

Jie-Yong Juang was born in Taipei, Taiwan, Republic of China, on January 10 1954. He received his B.S. degree in Electrical Engineering from Nationa Taiwan University, Taipei, Taiwan, in 1976, and his M.S. Degree in Compute Science from University of Nebraska, Lincoln, Nebraska, in 1981.

He is currently a Ph.D. candidate in the school of Electrical Engineering Purdue University. His research interests include computer architecture, com puter networks, distributed processing, and parallel processing.