Abstract

ARAS, ÇAĞLAN M. Communication Networks for Autonomous Mobile Robot Computer Architectures and Distributed Real-Time Computing. (under the direction of Dr. Ren C. Luo and Dr. Douglas S. Reeves).

Communication networks play an increasingly important role in many areas of computing. This dissertation introduces an interprocessor communication network for mobile robot computer architectures and real-time communication networks for distributed real-time computing. In both cases, the objective of the research is to provide communication with very low delay.

Mobile robot tasks such as collision avoidance and dynamic navigation change over time and exhibit a high degree of cooperation. The *segmented bus* is a reconfigurable interconnection network for intelligent mobile robot applications. The segmented bus employs a *preemptive circuit-switched* message transfer technique for low delay. For mobile robot applications, analysis suggests that message delays are lower on the segmented bus as compared to alternatives such as the multiple bus.

Interactive real-time message streams are used in continuous media, distributed sensing and command/control applications. These streams require low end-to-end delay bounds and zero-loss. This dissertation offers solutions to (i) message-stream admission, and (ii) low latency message transfers for interactive real-time message streams on communication networks. The proposed *adaptive stream admission* procedure is shown to admist more streams to a network than previous approaches. The proposed *preemptive cut-through* data transfer mechanism reduces the minimum achievable end-to-end delay bound of a packet-switched network to values approach-

ing circuit-switched techniques. To obtain low latencies on integrated broadband networks, the Asynchronous Transfer Mode (ATM) must support bound-delay, zero-loss data transfers. A *message-level* delay control mechanism based on preemptive cut-through is presented. Hardware and software to implement message-level delay control are described. Results show a significant reduction in computation requirements when message-level delay control is used on ATM switch processors.

Communication Networks for Autonomous Mobile Robot Computer Architectures and Distributed Real-Time Computing

by

Çağlan M. Aras

A thesis submitted to the Graduate Faculty of
North Carolina State University at Raleigh
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

Raleigh

1992

Approved By:

Chair of Advisory Committee	Co-Chair of Advisory Committee

to Cappy, for all the laughter...

Acknowledgments

First and foremost, I must express my gratitude to my advisors Dr. Ren C. Luo and Dr. Douglas S. Reeves. They both gave invaluable guidance and support throughout my Ph.D. studies. Dr. Luo put his trust in me by letting me direct the MARGE project; this has been one of the most rewarding experiences of my life. Dr. Reeves was my devil's advocate. His constant barrage of constructive criticism has hopefully taught me the fine points of critical investigation and clear writing. I also thank my committee members Dr. Harry Perros, Dr. Robert Fornaro and Dr. Edward Gehringer for their time and suggestions during the course of my research.

As the ancients say, the body is made of earth, but love feeds the soul. The camaraderie of the members of the robotics laboratory will be remembered for a long time. My particular thanks go to Dr. Mike Kay for his resourceful answers to my strange questions and Harsh Potlapalli for being a good buddy in the late hours of the night.

My parents, Dr. Namık K. Aras and Mrs. Çiğdem Aras have been behind me all the way through this endeavor and deserve special thanks. My Internet friends have supported me from as far-away places as Lausanne, Paris, Vancouver and Ankara; I'm glad you're out there.

I reserve the final thanks to my sister Pınar. In my last year, she provided constant encouragement, nightly dessert breaks and the laughter to keep me going through the tough times.

Table of Contents

List of Figures vi

List of Tables ix

List of Symbols xi

1 Introduction 1

1.1 Interconnection Networks for Mobile Robot Computer Architectures . 2

1.2 Real-Time Communications 5

1.3 Overview of the Dissertation 9

2 The Segmented Bus 11

2.1 Previous Work 13

2.2 Message Transfers on the Segmented Bus 17

2.2.1 Preemptive Circuit-Switching 17

2.2.2 Virtual Cut-Through 19

2.3 Performance Analysis of the Segmented Bus 19

2.3.1 Computing Loads in Segments 19
2.3.2 Uniform Message Distance Probability Distribution 21
2.3.3 Decaying Message Distance Probability Distribution 22
2.3.4 Simulation 25
2.4 Results 27
2.5 Implementation of the Segmented Bus 39
2.5.1 Data Transfer Operations on the Segmented Bus 40
2.5.2 Buscon 40
2.5.3 Gate 42
2.5.4 Arbus 46
2.6 Discussion 47

3 Real-Time Communication Networks **49**

3.1 Low Latency Message Transmission 53
3.2 Implementation of Real-time Message Services 56
3.3 High Availability Network Resource Allocation 57

4 Low Latency, Low Jitter Message Transmission **59**

4.1 Arrival Over Time 60
4.2 Preemptive Cut-Through 62
4.3 δ-stamping 64
4.4 Real-Time Message Transfers with ATM 67

4.4.1 Message Level QOS Maintenance in ATM 70
4.4.2 The Preemptive Cut-Through Scheduler 75
4.4.3 Processing Cut-Through Timing 78
4.4.4 Performance Analysis of Message-Level Delay Control 79
4.5 Discussion 83

5 Adaptive Message Stream Admission **86**

5.1 Adaptive Stream Admission 89
5.2 Performance Analysis of Adaptive Stream Admission 96
5.3 Discussion 99

6 Conclusions **102**

6.1 Contributions of Research 102
6.2 Suggestions for Future Work 105

Bibliography **107**

A MARGE: Mobile Autonomous Robot for Guidance Experiments **113**

List of Figures

1.1 Process graph for the *roam* mode 3

1.2 Process graph for the *inspection* mode 4

2.1 The segmented bus 11

2.2 Example for preemptive circuit-switched message transfers 18

2.3 Components of effective arrival rate λ_j^* 20

2.4 $\phi(\gamma, H_{max}, H)$ vs. γ for H_{max}=9, H =0...9, $\gamma = 0.1 \ldots 0.6$ 23

2.5 Aggregate loads per segment λ_i^*, relative to λ vs. γ 25

2.6 Average delay (μsec) vs. γ for multiple bus, virtual cut-through and preemptive circuit-switched message transfers 30

2.7 Average latencies (μsec) vs γ for virtual cut-through and preemptive circuit-switched message transfers with ρ=0.2 ...0.5 34

2.8 Virtual cut-through message transfer delays for $\gamma = 0.1$ 35

2.9 Preemptive circuit-switched message transfer delays for $\gamma = 0.1$ . . . 36

2.10 Virtual cut-through message transfer delays for $\gamma = 0.3$ 37

2.11 Preemptive circuit-switched message transfer delays for $\gamma = 0.3$ . . . 38

2.12 Segmented bus hardware 39

2.13 Gate hardware organization 43

2.14 Virtual cut-through and preemptive circuit-switched message transfer state diagrams 45

3.1 Example network 51

3.2 Deadline-based scheduler parameters 54

4.1 Total end-to-end latency 62

4.2 δ-Stamping 65

4.3 ATM QOS maintenance mechanisms 68

4.4 A preemptive cut-through link manager for an ATM switch 73

4.5 Message detector state diagram 75

4.6 Message storage hierarchy for preemptive cut-through 76

4.7 Processing delay overlap 83

5.1 Adaptive stream admission procedure 95

5.2 Test network for adaptive stream admission (Adapted from [1]) . . . 96

5.3 Adaptive vs. non-adaptive stream admissions 100

5.4 Histogram for adaptive vs. non-adaptive stream admissions—$\psi = [0.1, 0.6]$ 100

A.1 The MARGE vehicle 114

A.2 The MARGE computer system 116

List of Tables

1.1 Quality of service requirements for real-time applications 7

2.1 Notation used for the segmented bus 12

2.2 Previous work in bus-based and reconfigurable interconnection networks 15

2.3 Weighed average of latencies for virtual cut-through transfers 29

2.4 Weighed average of latencies for preemptive circuit-switched transfers 30

2.5 Multiple bus message transfer latencies 31

2.6 Comparison of simulation and analytical method for effective (ρ) confidence intervals 33

3.1 Notation for real-time message streams 50

3.2 Real-time properties of previous methods 51

3.3 Notation for link operation parameters 54

4.1 Delays and operations performed on an arriving cell 80

4.2 Performance comparison of cell and message-level delay control 81

4.3 p_{cell} vs. p_{msg} for various queue lengths 83

5.1 Stream admission procedure, adapted from [2] 87

5.2 Stream admissions for selected ψ ranges 98

5.3 Stream admissions for $\psi = [0.1, 0.6]$ 99

A.1 Contributors to the MARGE project 119

List of Symbols

Segmented Bus

PCS	Preemptive circuit-switched
VCT	Virtual cut-through
MBUS	Multiple bus
SBL	Segment Boundary Left
SBR	Segment Boundary Right
H_i	Number of hops (message distance) for m_i
N	Number of segments on a segmented bus
P	Total number of processors
P_i	Processor i
Γ_i	Path (set of segments) for m_i
Ω^j	Message being carried on s^j
Π^j	Set of processors in s^j
Π^j_H	Highest processor id in s^j
Π^j_L	Lowest processor id in s^j
γ	Outgoing message factor

λ_j	Arrival rate of data generated in s^j
λ_j^*	Effective arrival rate to s^j
μ	Bus service rate
ϕ	Decaying message distance probability distribution
π_i	Priority of m_i
ρ	Utilization at inputs of interconnection network
τ_i	Transmission duration of m_i
m_i	Message i
$m_s(j)$	maximum reach for s^j
$o(i)$	Processor originating M_i
p_{jk}	Probability of transfer for m_i from s^j to s^k
$q_s(j)$	Symmetric reach for s^j
$r(i)$	Processor receiving M_i
s^j	Segment j

ASA	Adaptive stream admission
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband-Integrated Services Digital Network
DBR	Delay bound reduction
EDF	Earliest deadline first
ISDN	Integrated Services Digital Network
PCT	Preemptive cut-through
QOS	Quality of service
RT	Real-Time
BA	Number of operations to allocate buffer space
BC	Duration of cell copy to buffer location
DS	Duration of delta stamp field arrival
HA	Duration of cell header arrival
I	Number of operations per queue entry traversal
PQ	Total number of prequeueing operations
Q	Number of operations to insert message record into a priority queue
TL	Number of operations to lookup VCI/VPI.
TS	Number of operations to compute arrival time
D_i'	Network offered maximum source to destination delay
D_i	Maximum source to destination delay
D_{cell}	Maximum source to destination delay with cell-level control
D_{msg}	Maximum source to destination delay with message-level control

$F_i(t)$	Full period count of M_i at time t
H_i	Number of hops on path ($\|\Gamma_i\|$)
J_i	End–to–end delay jitter
$L^j(t)$	Total usage time of ℓ^j in $[0,t]$
M_i	Message stream i
$R_i(t)$	Residual period count of M_i at time t
S_i	End–to–end slack
T_i	Period
a_i^j	Time whole of m_i is transmitted on ℓ^j
d_i^j	Delay bound for m_i on ℓ^j
$d_i'^j$	Minimum delay bound for m_i on ℓ^j
$\tilde{d}_i^j$	Expanded delay bound for m_i on ℓ^j
h_i^j	Time header of m_i arrives to node via ℓ^j
ℓ^j	Link j
m_i	A message in stream M_i
n	number of cells per message
p_{cell}	Processing delay with cell-level control
p_{msg}	Processing delay with message-level control
q	Number of messages per queue
r_i^j	Time a message is ready to be sent on ℓ^j
t	time
t^j	Time on link j
t_{fi}	Starting time of a forced idle
w_i^j	Link waiting time $w_i^j = d_i^j - \tau_i^j$ on ℓ^j

z_i^j	Message deadline $z_i^j = r_i^j + d_i^j$ on ℓ^j
Γ_i	Path (set of links) for M_i
Ω^j	Set of streams served by ℓ^j
α_i^j	Maximum value by which d_i^j can be expanded
β	Link transmission rate increase ratio
κ_i^j	Link slack
λ	Link arrival rate
μ	Link transmission rate
ψ	Delay bound ratio
ρ	link utilization
τ_i	Transmission time
ξ	processor speed multiplier

Chapter 1

Introduction

Advances in computers have continued to provide hardware to satisfy the ever increasing demand for more computational power. Multi-computers offer even more computing power to special applications by combining the resources of several computers through an interconnection network. Applications benefitting from multi-computers consist of many loosely coupled tasks. For example, in mobile robots, the tasks include vision, sensory processing and integration, and motion planning.

The processing requirements of tasks influence the architecture of multi-computers. Usually, the interprocessor communication network needs to be tailored to take advantage of application characteristics. This customization increases the performance of the multi-computer by balancing the use of computing and communication resources.

Distributed computing is similar to multi-computing in the sense that computers cooperate through a communication network. However, in distributed computing, the purpose of the communication network is to disseminate information for independent processing at each site. This increases the quality of information provided

by computers. With *Distributed Real-Time Computing*, the disseminated information is used by real-time applications. Examples of distributed real-time applications include command/control systems, remote sensing and monitoring, and digital video conferencing.

High-speed communication facilities are a prerequisite for distributed computing. For distributed real-time computing, the communication facilities also need to provide guarantees on the timeliness of the data transfer. Recent efforts have been directed toward collecting diverse communication networks into a single integrated network. The provision of real-time communication services over integrated networks is essential for distributed real-time computing applications.

1.1 Interconnection Networks for Mobile Robot Computer Architectures

The research to endow mobile robots with "intelligence" to has started in the early 1970's [3] and is still active today with projects such as a Mobile Van [4] and our own MARGE (Mobile Autonomous Robot for Guidance Experiments) project [5] in the Center for Robotics and Intelligent Machines at the North Carolina State University (See Appendix A for a description of MARGE).

Autonomous mobile robots, by virtue of their design, must be able to function in many different environments and perform a variety of tasks. Each task and environment poses different computational requirements. We illustrate this with an example. Consider a mobile robot with a video camera for vision, sonar and infrared sensors

for object detection, and odometric devices to measure traveled distance. When the mobile robot is roaming in its work environment, processes controlling navigational tasks are active: sonar, infrared and odometry processing, coupled with motion planning. Vision tasks search for landmarks but cannot perform detailed examinations due to distances and real-time navigation constraints. The process graph in the *roam* mode is shown in Figure 1.1.

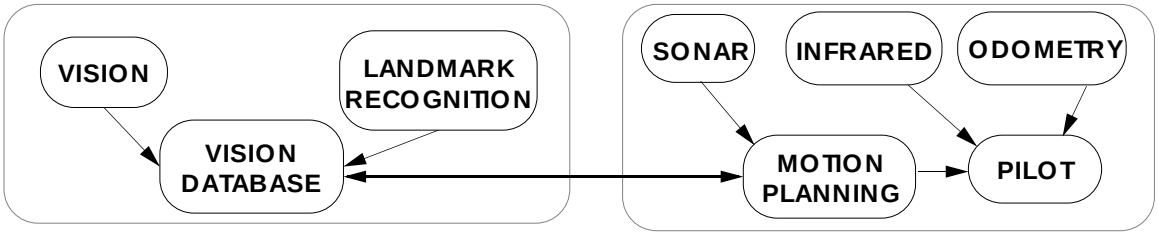


Figure 1.1: Process graph for the *roam* mode

When a goal point is reached, navigation is no longer required. Instead, more detailed vision processing takes over is to examine the goal point for a known landmark. In this case, the process graph changes to the *inspection* mode (Figure 1.2). This simple example illustrates two characteristics of the computational processes on a mobile robot:

- Tasks in intelligent robot control software are large but few [6]. The active processes forming a robotic task are grouped into cooperating *modules*. Intra-module communications are high due to the level of cooperation.
- The *process communication graph* changes over time as the robot engages in different activities.

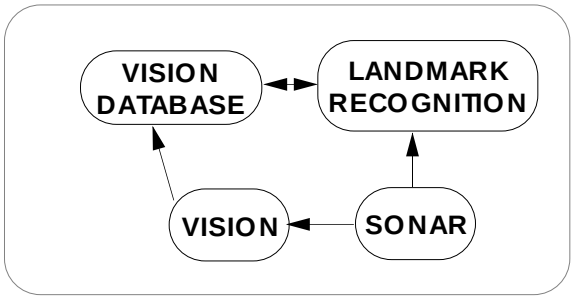


Figure 1.2: Process graph for the *inspection* mode

Mobile robots need computers that can execute large sets of cooperating processes with time varying communication demands. Multi-processors can potentially solve this problem, although the interconnections between processors are usually fixed. While task relocation can accommodate changes in the process communication graph, it is costly in terms of the required data transfers and operating system level changes.

Reconfigurable interconnection networks satisfy the needs of mobile robot tasks. They can be configured to group processors into *clusters*. As the process graphs change, cooperating tasks can be kept in the same cluster by reconfiguring the network.

The large scale reconfiguration of an interprocessor communication network has not received much attention in the literature. Static clustering has been examined in Cm* [7]. With static clustering, it is not possible to change cluster sizes. If a large group of tasks need to cooperate closely and are allocated to a single cluster, since the number of processors in each cluster is fixed, the processing time may be excessive. On the other hand, if the tasks are allocated to different clusters, the

communication time may be excessive. The lack of network flexibility prevents tuning communication bandwidth requirements. In contrast, enhancing the architecture to permit dynamic reconfiguration will improve the performance of the system for this type of application [8].

Buses are the dominant medium of interprocessor communications for multiprocessors for robot control systems, having highest bandwidth of all interprocessor communication media. Reconfigurable networks of buses have been mainly used in massively parallel computer systems. Li and Stout [9] provide many examples of such systems. The reconfiguration of buses has been studied in the context of the problems of arbitration for adjacent buses [10], however, as of yet, buses have not been used as a message-passing network for clustering purposes.

This work examines the feasibility of a new reconfigurable interprocessor communication network called the *segmented bus*. A new *preemptive circuit-switched* message transfer method is introduced. Preemptive circuit-switched reduces the inter-segment communication delays in multi-hop networks such as the segmented bus. The combination of the segmented bus and preemptive circuit-switched is compared to other bus-based interconnection networks.

1.2 Real-Time Communications

The formation of integrated wide area communication networks such as B-ISDN [11] was motivated by a desire to use a *single* network to provide communication services to a variety of service requests. These requests range from the traditional data-

gram and connection-oriented data services to the transmission of digitized video, audio and sensory information. Datagram and connection-oriented data communications are currently served by protocols such as TCP/IP and UDP. Applications using these protocols include e-mail, file transfers, transaction processing and remote file and computer access. The other service requests to be included in these integrated networks have traditionally been served by dedicated circuit-switched lines which inherently provide service in *real time*. Pay-per-view TV, telephone and leased lines for air-traffic control are typical examples of the real-time services envisioned to coexist with traditional data communications in an integrated network. A new class of multi-media applications such as video conferencing [12] and remote medical assistance combine data and audio-visual information. These applications require both data and real-time services and are expected to make significant use of integrated networks.

For a service request, the *Quality of Service (QOS)* defines acceptable levels for parameters such as *delay, data loss*, and *jitter*. In existing communication networks, the QOS is based on measures of performance for *average* delay or *average* data loss [13] only. Recent work has focused on QOS guarantees for individual packets within a packet-switched network. On the other hand, circuit-switched transfer methods can provide QOS guarantees for entire messages. In addition, circuit-switched networks can easily provide the qualities of service required by real-time applications, such as guaranteed bounds on end-to-end delay, zero message loss, and zero jitter. For integrated networks to be widely accepted, to use packet-switched networks for real-time applications, the networks must provide these same qualities of service. QOS enhancements such as the *synchronization* [14] of multiple real-time sources, which is

useful for multi-media applications, may be required as well.

In B-ISDN terminology, real-time message streams are labeled as Class A or *constant bit rate* data [15]. This means that the amount of data sent per unit time does not vary. On circuit-switched networks, all real-time message streams are transported within a low delay bound and without loss. Integrated networks such as B-ISDN can mix streams with different delay and loss requirements. Therefore, the delay and loss can be adjusted for each stream. For real-time streams, the delay/loss matrix for various applications is shown in table 1.1.

Table 1.1: Quality of service requirements for real-time applications

	Delay Sensitive	Delay Insensitive
No-Loss	Critical Applications: • Command/Control Systems • Remote Medical Assistance • Interactive Visualization	
Loss Tolerant	Interactive Applications: • Telephone • Video Conferencing	Playback Applications: • HiFi Audio Broadcasts • HDTV Broadcasts

The applications in table 1.1 are classified into three types:

Critical Applications: Command/control systems such as air traffic control require low delays and zero data loss [16]. In these applications, any delayed or lost data can result in a loss of life. When medical experts monitor surgery from a remote site, fast and accurate transmission of patient vital signs is essential. Interactive visualization depends on low delay and zero loss communication facilities for accurate rendering of high resolution graphics or animations.

Playback Applications: Digital high-fidelity audio and high definition television broadcasts fall in this category [17]. The messages in these streams must be delivered within a delay bound to limit data-jitter. However, the absolute delay need not be low; it is immaterial whether video feed from a station reaches the viewers in 2 milliseconds or 3 seconds. Playback streams can tolerate losses. For audio, up to 1-2% [18] loss is acceptable.

Interactive Applications: In telephone conversations and digital video conferencing two or more people interact with each other. In these applications, low message delays and low jitter are needed to maintain a "live" conversation. As in playback applications, a certain level of loss is tolerable.

The second major objective of this dissertation is the development of real-time communication techniques for *critical* real-time applications. A new real-time message transfer method called *preemptive cut-through (PCT)* is presented. This method reduces the lower bound on end–to–end delay to values seen in circuit-switched networks, while retaining the same qualities of qervice otherwise provided by previous real-time message transfer methods.

B-ISDN is still in the planning stages, and as a result many standards have not yet been finalized. However, ATM (Asynchronous Transfer Mode) [19] has been selected as the low-level transport mechanism. ATM is based on fast packet-switching of small (53 byte) cells in high speed switches (see [20] for an overview). At a conservative 1Gb/sec optical link bandwidth, there is only approximately 400ns of processing time for each cell. The qualities of service which are required for critical real-time message streams impose a significant overhead on each transmission. This overhead

will probably be excessive at a per-cell level. A *Message-level* QOS maintenance to reduce the overheads of cell-level QOS maintenance is presented and evaluated.

One of the important features of integrated networks [11] is the requirement for establishing a connection for each service request. This requirement is useful for real-time communications since the quality of service must be guaranteed before the message stream can be initiated. For real-time applications, the maximum end–to–end delay and the cell loss rate are the two most important QOS parameters. The term *message stream admission* refers to all of the actions associated with establishing a connection for real-time message delivery. The goal of any message stream admission technique is to admit the highest number of real-time message streams to the network. Therefore, the efficient allocation of network resources is of utmost importance. In the case of end-to-end delay based resource allocation, current methods for message stream admission [2, 21, 22] are overly simplified in bandwidth allocation. This dissertation presents a method of message-stream admission called *adaptive stream admission*. Adaptive stream admission is a dynamic method of bandwidth allocation which admits greater numbers of real-time message streams to a network.

1.3 Overview of the Dissertation

This dissertation is formed of two major parts. The first part describes the *segmented bus*, a proposed multi-computer interprocessor communication network for mobile robot computer architectures. In chapter 2 first the previous work related to reconfigurable interconnection networks are discussed. Dynamic segmentation and the proposed preemptive circuit-switched (PCS) message transfer technique are de-

scribed. The performance of preemptive circuit-switching is analyzed in section 2.3. The segmented bus is compared to the single busand the multiple busin terms of average message transfer delays. The preemptive circuit-switched message transfer technique is compared to virtual cut-through. The chapter concludes with an overview of segmented bus implementation issues.

Chapters 3–5 describe communication techniques for distributed real-time computing. Chapter 3 defines the system model and the problems to be solved. Chapter 4 introduces preemptive cut-through. In section 4.4, the implementation of preemptive cut-through within B-ISDN and ATM to provide message-level quality of service is described. Message-level quality of service maintenance is compared to cell-level maintenance in terms of computational power requirements and processing overheads. Chapter 5 defines and analyzes adaptive stream admission. This method is compared to other proposed message stream admission methods through an experiment.

Chapter 6 summarizes our research contributions and findings, and suggests future extensions of this research.

Chapter 2

The Segmented Bus

The segmented bus is a bus based interprocessor communication network. The network is formed of N buses called *segments*, that are linearly connected. Each segment j, denoted as s^j, is a fully functional bus. A *gate* connects two adjacent segments s^j and s^{j+1} to transfer data between segments (Figure 2.1).

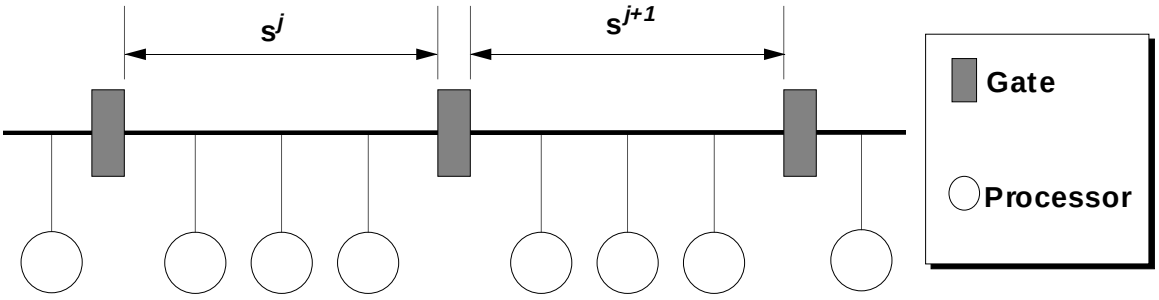


Figure 2.1: The segmented bus

Any data being passed around the segmented bus is destined to a specific processor. Processors are identified by their *processor identification number*, P_i. On the segmented bus processors are numbered $1 \ldots N$, from left to right. The segmentation

of the bus does not change the processor identification numbers. The set of processors in s^j are denoted by Π^j; $\Pi^j_i = \{P_i \in s^j\}$. In order to identify these processors, there are two bounding values Π^j_L and Π^j_U, denoting the lower and upper range of P_i values in s^j.

A message m_i on the segmented bus is represented by a tuple $(\tau_i, \Gamma_i, \pi_i)$. τ_i denotes the length of the message. Functions $o(i)$ and $r(i)$ return the originating and receiving processor of m_i respectively. For a message originating at processor $P_{o(i)} \in s^j$ and to be received by processor $P_{r(i)} \in s^j$, the path Γ_i denotes the segments $i, \ldots, j$ traversed by m_i. Each message also has a priority π_i which can be set statically at transmission time or which can vary with other measures such as a message delivery deadline.

At any time, the message traversing s^j is denoted by Ω^j. As an example, the priority of the message using s^j is π_{Ω^j}. The notation used is summarized in table 2.1.

Table 2.1: Notation used for the segmented bus

Symbol	Meaning
P	Total number of processors
N	Number of segments
s^j	Segment j
Π^j	Set of processors in s^j
Π^j_L	Lowest processor id in s^j
Π^j_H	Highest processor id in s^j
m_i	Message i
τ_i	Transmission time of m_i
$o(i)$	Processor originating m_i
$r(i)$	Processor receiving m_i
Γ_i	Path (segments used) by m_i
π_i	Priority of m_i
Ω^j	Message being carried on s^j
λ_j	Arrival rate of data generated in s^j
λ^*_j	Effective arrival rate to s^j
p_{jk}	Probability of data going from s^j to s^k

The segmented bus is intended to match characteristics of robotic tasks. Each s^j can collect processors executing cooperating processes to one segment. This provides *clustering*. The segments in the segmented bus can be dynamically created, removed or resized by activating, removing or changing the position of gates. This provides the ability to tailor the network to changes in the processing graph.

In this chapter, the benefits of the segmented bus are examined. The following problems are addressed:

- What techniques can be used to transfer data within and across segments?
- Does the segmented bus offer improved performance?
- How can a bus be dynamically segmented? What hardware mechanisms and software protocols are necessary?

2.1 Previous Work

To compare the segmentation and clustering abilities of interconnection networks, we define the *reach notation* below.

Definition 2.1 (Hop Cost) *In a multi-hop network, the* hop cost H *is the number of intermediate nodes or gateways traversed between the source and destination of a message. For* m_i, $H = |\Gamma_i|$.

Definition 2.2 (Global Reach) *The* Global Reach $GR(H)$ *is the number of processors that can be reached from a processor with a hop cost* H.

Definition 2.3 (Local Reach) *The* Local Reach $LR = GR(0)$ *is the number of processors reachable without traversing any gateways or intermediate nodes.*

The reach notation can be used in task assignments. The global reach measure is similar to the *dilation* measure [23]; however, the global reach provides more information about the interprocessor communication network than its dilation. Since the dilation of tasks have been used for optimum task assignment, the suitability of an interconnection network to task assignment can be examined by using its global reach set parameters as well.

To clarify the reach notation, suppose a multi-computer has P processors. For the single bus, $LR = P$. With a two-level hierarchical bus, LR includes the processors within a cluster. The one hop reach $GR(1)$ includes all processors in clusters that can be reached with one hop. For a hierarchical bus structure such as Cm* [7], LR includes all processors reachable through the map bus and $GR(2)$ includes all processors in the system reachable by the inter-cluster bus.

Using the reach notation, various bus-based interprocessor communication networks are compared in table 2.2. In the table, P denotes the total number of processors served by the network. If the network can cluster processors or segment the interconnection network, N denotes the number of clusters or segments. As can be seen from the table, only the DS Bus [10] approaches the flexibility of the segmented bus. However, the DS Bus does not provide any buffering between segments. Other solutions [24, 25] either limit the cluster size or do not allow inter-cluster transfers. Others [7, 26] are not reconfigurable. The bus based solutions such as the single and multiple bus [27, 28] hold potential; their effectiveness is compared with the segmented

bus in this work.

Table 2.2: Previous work in bus-based and reconfigurable interconnection networks

Method	LR	GR	Limitations
Single Bus	P	-	Low fan-out limits expansion
Multiple Bus[27]	P	-	High interfacing cost and limited expansion
Partial M-Bus[28]	P	-	High interfacing cost and limited expansion
Cm*[7]	fixed	$GR(3) = P$	Fixed cluster size, no dynamic reconfiguration
MP/C[24]	$[1, P]$	not possible	No inter-segment transfers
HM[25]	3	$GR(1) = P$	Small clusters, slow inter-cluster transfers
Multi-Segment[26]	1	$GR(H) = 2H$	No dynamic reconfiguration. Very small cluster size limits cooperating processes
DS Bus[10]	$\frac{P}{N}$	$GR(H) = \frac{P}{N}H$	No data buffering for inter-segment transfers
DS Ethernet[29]	$\frac{P}{N}$	$GR(H) = \frac{P}{N}H$	No inter-segment transfers

The connection of buses and data transfers across bus segments are key to implementing dynamic segmentation. The issue of bus connection was first addressed in the MP/C architecture [24]. The Gated Connection Network [30] has provided an implementation of a bi-directional transmission gate circuit allowing the connection and isolation of data lines. A precharge circuit is used to minimize switching delays. This circuit is for single serial lines, but could also be used for parallel signals on a bus.

A more limited case of bus connection has been proposed in the Homogeneous Processor [25]. In this architecture, a bus switching mechanism is used to connect two adjacent processors. The hardware protocol assures that only one processor can

access the memory while the other processor is blocked. The connection time is limited to one memory access. In a more recent publication, Athas [26] describes a multi-segment bus architecture with only one processor per segment. Each segment boundary allows data transfers using wormholing and multiple data transfers may be established concurrently.

Multiple-Bus architectures [27] have been proposed as a replacement for the single bus since they provide increased bandwidth using duplicated bus hardware. These architectures have been aimed at shared memory systems. They can potentially be used as a message-passing interconnection network, as described by Ganz [31]. With the multiple bus, to obtain meaningful gains in bandwidth, large numbers of parallel buses need to be used, increasing hardware costs. To reduce hardware costs, partially connected multiple bus architectures have been proposed by Lang [28].

Techniques for task allocation, re-mapping and allocation of tasks onto reconfigurable systems are required for the effective use of a reconfigurable architecture. These techniques have received increased interest with the availability of multiprocessor systems. Chu [32] has presented one of the earlier works in this field while Lo [33] and Houstis [34] present newer approaches to static task assignment. Since these algorithms are aimed at large scale task allocation, the assignment of individual tasks requires a total re-evaluation of the task communication graph. For re-mapping of tasks, the issue of relocation has been investigated by Nicol [35] for varying demand distribution problems, which are task groups with varying computation demands; the intelligent robot control problem falls in this domain. Distante [36] and Du [37] provide algorithms for allocating tasks onto reconfigurable architectures. The cur-

rent state of research in task allocation does not provide solutions to the dynamic re-mapping of architectures based on task activity. However, these solutions can still be used if changes in task computational requirements are somewhat infrequent.

2.2 Message Transfers on the Segmented Bus

The segmented bus is similar to a single bus when data is transferred within a segment. Since segments are isolated, processes in different segments communicate by crossing segment boundaries. This is called an *inter-segment transfer*. Although inter-segment data transfers are not expected to occur frequently, they are still needed. The delay due to inter-segment data transfers is expected to be higher than intra-segment transfers. We propose a *preemptive circuit-switched* data transfer technique to limit inter-segment data transfer delays.

2.2.1 Preemptive Circuit-Switching

The preemptive circuit-switched (PCS) message transfer technique combines circuit-switching and preemptive priorities. As in circuit-switching, the whole path Γ_i must be set up before the transfer can proceed. For a new message M_k, all $s^j \in \Gamma_k$ are requested from an arbiter. Priorities determine the availability of a path. The path is available if all $s^j \in \Gamma_k$ are free or if $\forall\ m_i : \Gamma_i \cap \Gamma_k \neq \emptyset$, $\pi_k > \pi_i$. In this case, all $m_i : \Gamma_i \cap \Gamma_k \neq \emptyset$ with $\pi_k > \pi_i$ are preempted. The preempted message transfers are resumed when segments become free or when their priority is higher than pending requests for the segments they require.

The priorities for the preemptive circuit-switched transfer technique can be arbitrarily set depending on the system parameters of concern. For example, if there are periodic messages, Rate Monotonic Scheduling [38] can be used for real-time message transfers. For other deadline-based systems, the least slack algorithm [39] can be used as well. In this work, priorities were set based on the distance of the message path, i.e., $\pi_i = |\Gamma_i|$ with longer distances having higher priorities. The motivation behind this was to obtain near-uniform data transfer times for all distances.

A simple example illustrates preemptive circuit switching. Assume a message m_1 with parameters $(\tau_i, \Gamma_i, \pi_i) = (\tau_1, \{1, 2\}, 1)$ is transferred on a segmented bus with $N = 4$ segments (Figure 2.2). Suppose a new message transfer $m_2 = (\tau_2, \{4, 3, 2\}, 2)$ is requested. Since $\pi_2 > \pi_1$, m_1 is preempted in s^2. It is resumed after m_2 is transferred.

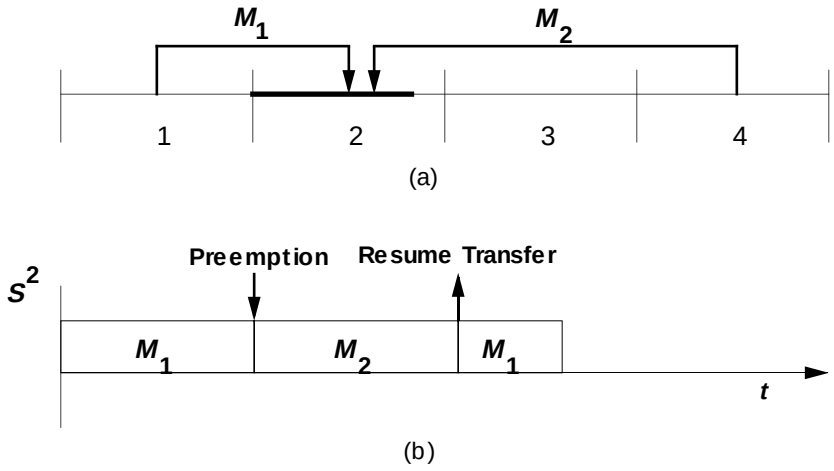


Figure 2.2: (a) Preemptive circuit-switched message transfer (b) View of message transfer on s^j

2.2.2 Virtual Cut-Through

The virtual cut-through (VCT) technique, first proposed by Kermani and Kleinrock [40], has been used in more recent interprocessor communication networks [41]. In [40] cut-through is only allowed if the next link in the path of the message is free at the instant the header is decoded. In later works, partial cut-throughs are also allowed, starting message transmissions any time the next link becomes free.

Virtual cut-through also provides low delays for inter-segment message transfers. We use virtual cut-through as a non-preemptive data transfer technique on the segmented bus.

2.3 Performance Analysis of the Segmented Bus

The performance of the segmented bus was evaluated using analytical techniques and simulation. The quantities analyzed were the effects of segmentation on the utilization of each segment and the delay performance of segmented bus. The average delay performance was compared to the multiple bus. The preemptive circuit-switched and virtual cut-through data transfer techniques were compared to examine the effects of preemption and message distance on delay.

2.3.1 Computing Loads in Segments

Processors Π^j on each s^j generate data at the rate of λ_j messages/sec. The *effective* arrival rate λ_j^* to a s^j is dependent upon the probability distribution of the message distances H.

Given a segmented bus with N segments, the probability of data generated in s^j remaining in the segment is denoted by p_{jj} and going to s^k is denoted by p_{jk} where $j, k \leq N$. For any s^j, $\sum_{k=1}^{N} p_{jk} = 1$.

For a transfer from s^j to s^k, m_i will originate in segment s^j, traverse all segments $j+1$ to k and arrive to a processor in s^k. Thus all $s^j \ldots s^k$ will be utilized. For s^j, the effective arrival rate λ_j^* can be broken down (Figure 2.3) into the following components:

1. Local data λ_j.
2. Data arriving to s^j from lower segments s^l, $l < j$.
3. Data from lower segments s^l, $l < j$ destined to higher segments s^k, $k > j$.
4. Data arriving to s^j from higher segments s^k, $k > j$.
5. Data from higher segments s^k, $k > j$ destined to lower segments s^l, $l < j$.

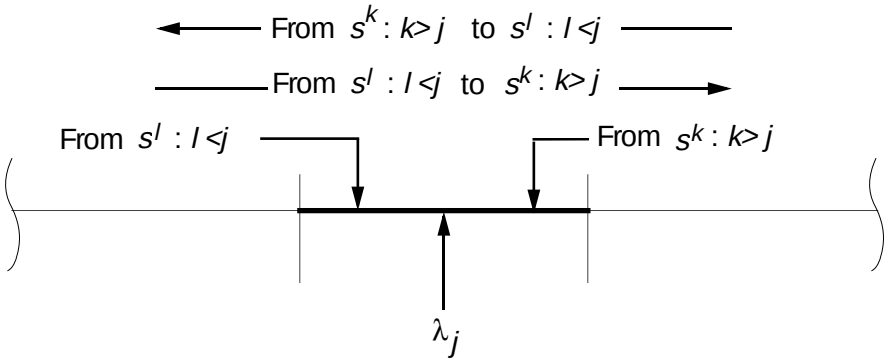


Figure 2.3: Components of effective arrival rate λ_j^*

Thus,

$$\lambda_j^* = \lambda_j + \sum_{i=1}^{j-1} \sum_{k=j}^{N} \lambda_i p_{ik} + \sum_{i=j+1}^{N} \sum_{k=1}^{j} \lambda_i p_{ik} \tag{2.1}$$

Since the purpose of creating segments is to isolate data flows, it is possible to assume that tasks are allocated to equate loads in each segment. This assumption equates all λ_j to a single value $\overline{\lambda}$.

2.3.2 Uniform Message Distance Probability Distribution

One of the advantages of a bus is its invariance to the location of the sender and receiver of the message, or in general, the load distribution of the network. Since the uniform message distance distribution is the least advantageous distribution for the segmented bus, we analyze this case first. For uniformly distributed message routing, all $p_{jk} = \frac{1}{N}$. Substituting this into (2.1),

$$\lambda_j^* = \overline{\lambda} + \overline{\lambda} \left[\sum_{n=1}^{j-1} \sum_{k=j}^{N} \frac{1}{N} + \sum_{n=j+1}^{N} \sum_{k=1}^{j} \frac{1}{N} \right] \tag{2.2}$$

Solving for s^j,

$$\lambda_i^* = \overline{\lambda} \left[2j - \frac{2j^2}{N} + \frac{2j}{N} - \frac{1}{N} \right] \tag{2.3}$$

Some interesting results from (2.3) are for the boundary cases. For s^1 and s^N, $\lambda^* = \overline{\lambda} \left(2 - \frac{1}{N}\right)$. This simple result can be intuitively explained. Since the load distribution is uniform, each of the $N-1$ segments will send $\frac{1}{N}$ of their load to this segment. Adding the local data yields $\lambda^* = \overline{\lambda} \left(1 + \frac{N-1}{N}\right)$ load, which reduces to the

result above. This is the minimum load on the segments. The maximum load can be found by differentiating (2.3) with respect to j. The maximum is in segment $\frac{N}{2}$ with load $\lambda^*_{N/2} = \overline{\lambda}\left(\frac{N+2}{2} - \frac{1}{N}\right)$ This is approximately $\overline{\lambda}\frac{N}{2}$ for large N, which is half the load on a single bus. These results confirm the observations published by Athas [26].

2.3.3 Decaying Message Distance Probability Distribution

As stated previously, the purpose of using a segmented bus is to take advantage of the high degree of cooperation amongst tasks by placing them in the same segment. Reed and Grunwald [42] have used a decaying probability function, $\phi(d, l, d_{max}) = DNORM(d, d_{max})d^l$ for the probabilities of message source to destination distances distances, where $DNORM$ is a normalizing constant. Substituting our own notation and modifying it to include the case of $H = 0$:

$$\phi(\gamma, H, H_{max}) = DNORM(\gamma, H_{max})\gamma^{H+1}$$

Figure 2.4 shows the probability distribution $\phi(\gamma, H, H_{max})$ of distances H vs the outgoing message factor γ. We formally define γ below.

Definition 2.4 (Outgoing Message Factor) *The* outgoing message factor γ *is the percentage of data leaving a segment.*

Definition 2.5 (Localization Factor) *The* localization factor $(1 - \gamma)$ *is the approximate percentage of data destined to processors within the same segment.*

In the segmented bus, the maximum number of segments traversed by each mes-

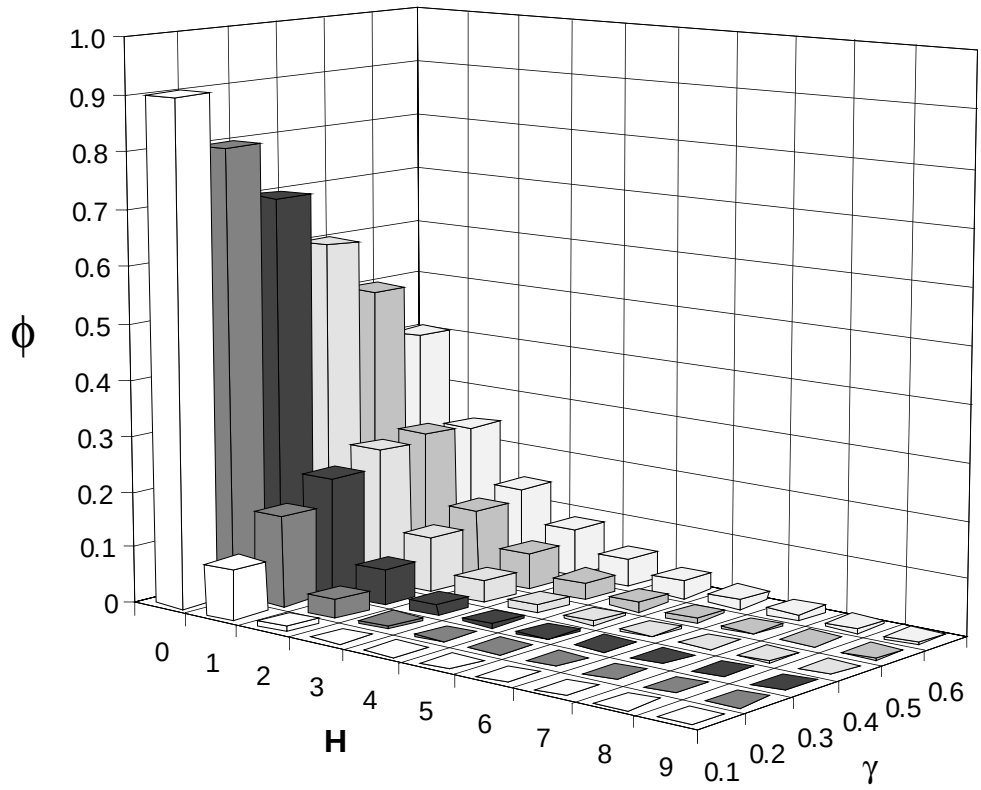


Figure 2.4: $\phi(\gamma, H_{max}, H)$ vs. γ for H_{max}=9, H =0...9, $\gamma = 0.1 \ldots 0.6$

sage is dependent on the location of the originating processor. We define the reach distances as follows:

Definition 2.6 (Maximum Reach) *The* maximum reach m_s *of a segment is the largest number of segments traversed by a message originating at* s^j.

This value is binomially distributed between $\left[\frac{N}{2}, N\right]$. For s^j, $m_s(j) = \max(N - j, j - 1)$. The maximum reach is only possible in one direction.

Definition 2.7 (Symmetric Reach) *The maximum reach in both directions is denoted by the symmetric reach* q_s. *This value has a maximum of* $\frac{N}{2}$. *For a* s^j, $q_s(j) = \min(N-j, j-1)$.

Since each segment has different maximum and symmetric reach values, the probability distribution for each segment is calculated separately. For s^j, the probability of sending messages to s^k is

$$p_{jk} = \begin{cases} j = k & \phi(\gamma, 0, m_s(j)) \\ |j-k| \leq q_s(j) & \phi(\gamma, |j-k|, m_s(j))/2 \\ |j-k| > q_s(j) & \phi(\gamma, |j-k|, m_s(j)) \end{cases} \tag{2.4}$$

Substituting (2.4) into (2.1) various results can be obtained for different values of γ. For $N = 10$, given a total network arrival rate λ and $\overline{\lambda} = \frac{\lambda}{N}$, the bus loads $\lambda_j^* \forall\ j \in [0, N]$, are plotted in figure 2.5 for various γ as well as the uniform distribution.

Figure 2.5 shows that for $\gamma \leq 0.3$, λ_i^* is about 15% of λ. On a single bus this value is 100% of λ. In other words, almost 7 times more bandwidth is available than is the case for the single bus. For $\gamma = 0.6$ (40% data locality), the $\max\{\lambda_i^* : 0 \leq i \leq N\}$ is only 29% of the single bus load. Even the uniformly distributed messages benefit from the segmentation of the bus. The values in the figure are valid for any segmented bus regardless of the data transfer mechanism used, showing the advantages of segmentation.

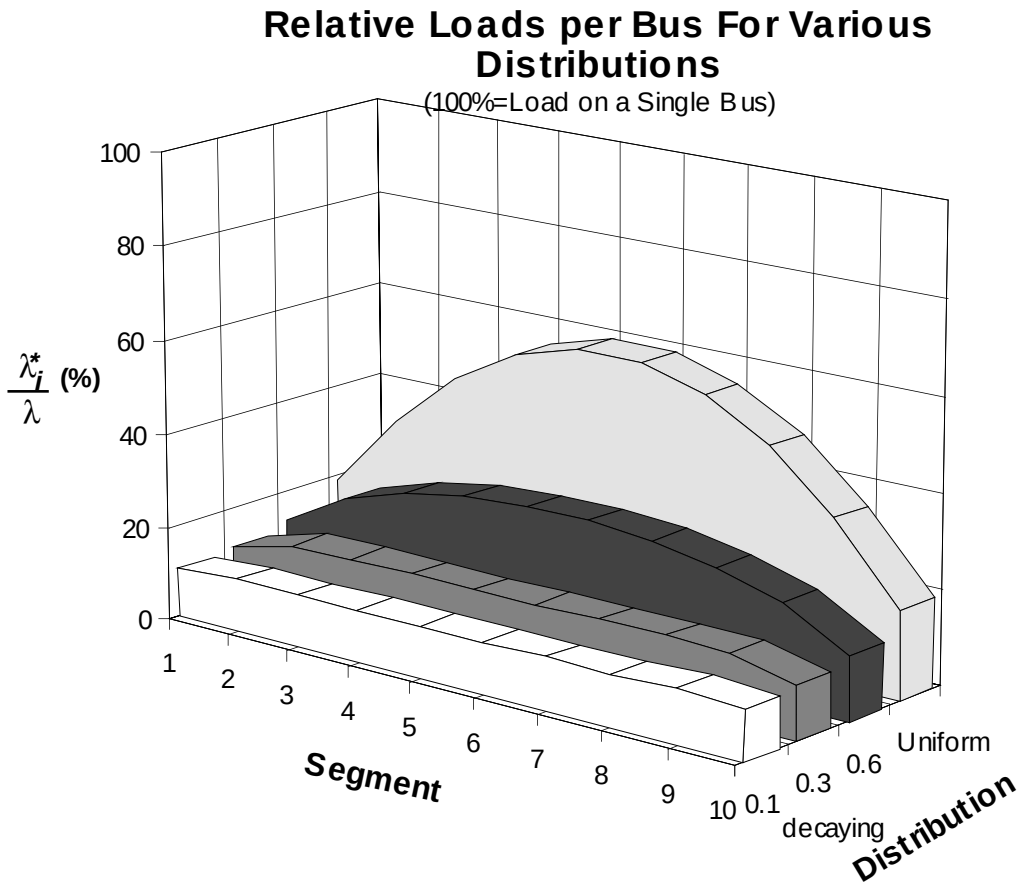


Figure 2.5: Aggregate loads per segment λ^*, relative to λ vs. γ, on a segmented bus with $N = 10$. A single bus bears the total load λ so its load is 100%.

2.3.4 Simulation

Simulation was used to obtain bounds on the latency and throughput of the segment bus with the virtual cut-through (VCT) and preemptive circuit-switched (PCS) message transfer techniques on the segmented bus interprocessor communication network.

The multiple bus was also evaluated as a comparison case. The multiple bus is offered as an alternative to the single bus and is similar to the segmented bus in

terms of the channel capacities of the buses. Since multiple bus architectures with a limited number of buses are realizable, they are an alternative to the segmented bus as well and should be examined. The analytical model presented by Ganz and Chalamtac [31] was used to evaluate the multiple bus. This model assumes a distributed memory architecture, which is similar to the segmented bus. In the Ganz model messages cannot pile up at nodes; aprocessor does not generate a new message until the current message is sent. This reduces the number of messages introduced to the interconnection network. Without this restriction, the latency estimates should be higher.

The SLAM II simulation language [43] was used for short simulation runs. For longer runs, a simulation program was written in C. The architectural model was based on a segmented bus with $N = 10$ and four processors per segment. Poisson interarrival times and exponentially distributed service rates were used; for large systems, these distributions have been found to be good approximations of message characteristics and fall within generally practiced guidelines.

The network arrival rates were varied from λ=20000 to 50000 messages/sec. For each bus, the transfer capacity was set to μ=10000 message/sec/bus. If all message transfers remained in the originating segment, this would be equivalent to ρ =20% to 50% for the network. In all results, these ρ values will be used. Aggregate network utilizations are higher due to messages using multiple segments during a transfer. With the selected value for μ, the results presented are in microseconds. As a reference value, the ideal transfer time with the given μ is 100 μsec.

To obtain realistic estimates, a 5% overhead was added to each transfer time to

incorporate either the message header or path setup times. The overhead for virtual cut-through transfers results from the message header reception time, while for the preemptive circuit-switched transfers, the preemption and path setup time are the cause for the overhead.

The decaying probability function $\phi(\gamma, H, H_{max})$ was used to assign the distance traversed for each message. The γ factor was varied from 0.1 to 0.3. For each γ, 30 runs of 100000 messages were used to obtain 95% confidence intervals.

The simulation runs have also yielded segment utilization values for each type of transfer. Some sample values are presented in Table 2.6.

2.4 Results

Simulation results have been obtained for the segmented bus for the two data transfer techniques and the multiple bus for three to five buses. Since the multiple bus (MBUS) is offered as a viable alternative to the single bus, first these results will be compared.

The results in tables 2.3-2.5 and figure 2.6 present the average delays for message transfers for virtual cut-through, preemptive circuit-switched and multiple bus transfers. The multiple bus and the segmented bus can only be compared on the average delay for all message distances since the concept of a message distance does not exist for the multiple bus. The results indicate that for most localized loads, ($\gamma = 0.1 \ldots 0.3$) and ρ=0.2, the segmented bus compares well with the multiple bus with three buses. For example, for ρ=0.2 and γ=0.2, the average delay for the multiple bus is 152 μsec while for virtual cut-through it is 145 μsec and for preemptive

circuit-switched it is 146 μsec.

Higher loads increase latencies for both systems and at some points exceed the capacity of the interconnection network. For example, for ρ=0.3, the multiple bus with three buses is saturated while the segmented bus can still handle the loads and offer a similar performance to the multiple bus with four buses for γ=0.2. Higher γ with decreased localization, or more buses result in lesser latencies for the multiple bus than the segmented bus.

There is a crossover point for which the performance of the multiple bus is less than the segmented bus. For $\rho = 0.2$, $\gamma = 0.3$ is where the multiple bus results become more favorable. For ρ=0.3, the crossover point is between γ=0.1 and 0.2. Thus, at increasing system loads, the locality factor plays an important role on the latency results.

The cause for the crossover is due to the two different effects observed in the two interconnection networks. In the segmented bus a significant source of delays is due to the multiplicative effect of the increase of γ on the total system load. At low γ, the localization of data transfers reduces λ^*, as discussed in Section 2.3. As γ is increased from 0.1 to 0.3, the λ^* in most segments increases by 50%, adding a significant amount of message traffic to most segments. So any increase in the total system load is in effect, multiplied by this factor. This effect decreases the *effective* bandwidth of the segmented bus, making it less than the multiple bus and therefore increasing the latencies to values higher than all multiple bus cases. It must be noted that high values of $\gamma > 0.3$ are not normally expected for the intelligent system control applications targeted for this architecture.

The multiple bus has several factors that result in higher latencies for low loads. Although each processor is connected to more than one bus, a processor can only transmit on one bus at a time. At low transfer rates, the per bus load on the multiple bus is higher than the loads on each segment of the segmented bus. As loads increase, the per bus load increase on the multiple bus is less than the increase on the segmented bus, (due to the multiplicative effects) and the crossover point is obtained.

It should be noted that the multiple bus system being examined may not be feasible. To be able to set up an equivalent system, it was necessary to place 40 processors on the multiple bus. Current technological constraints limit the number of processors to about 20 units. With less processors and the same load for the interconnection network, the effects of processor contention become the dominant factor in the delays. For example, the average delay for a multiple bus with 10 processors, five buses and the same system load ($\rho = 0.4$) as the systems listed in table 2.5 is 486 μsec, almost three times as much as the 40 processor cases we have examined. Therefore, some of the favorable latencies for the multiple bus must be considered with the limitations imposed by technological constraints.

Table 2.3: Weighed average of latencies for virtual cut-through transfers (μsec). All values are ± 1

	γ					
ρ	0.1	0.2	0.3	0.4	0.5	0.6
0.20	133	149	165	187	232	317
0.25	142	155	175	208	272	429
0.30	155	173	202	255	376	832
0.40	188	223	286	433		
0.50	238	308	467			

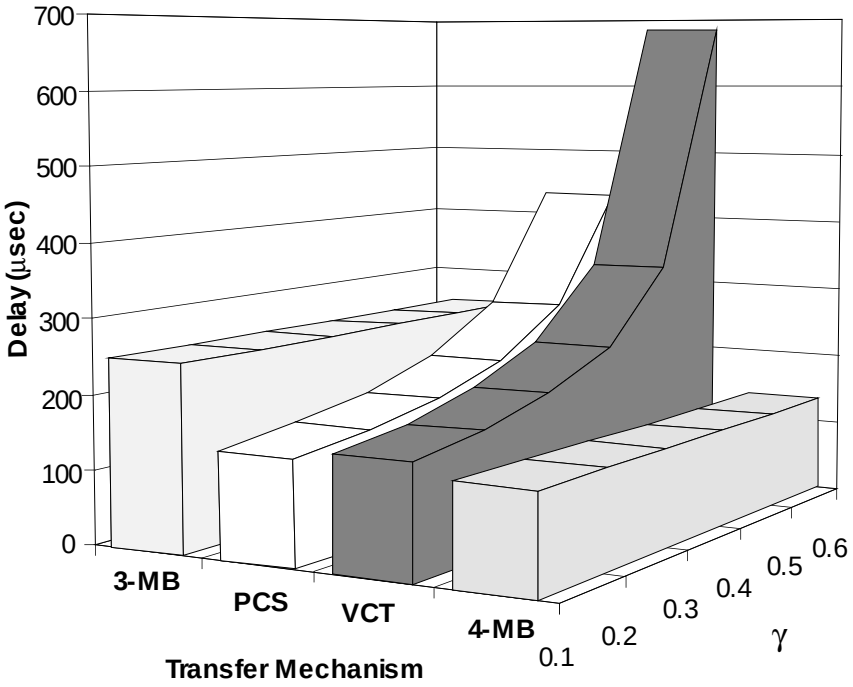


Figure 2.6: Average delay (μsec) vs. γ for multiple bus, virtual cut-through and preemptive circuit-switched message transfers with $\rho = 0.20$

Table 2.4: Weighed average of latencies for preemptive circuit-switched transfers (μsec). All values are ± 1

	γ					
ρ	0.1	0.2	0.3	0.4	0.5	0.6
0.20	133	146	163	191	241	335
0.25	151	167	193	239	336	685
0.30	164	187	226	303	518	
0.40	200	245	337	608		
0.50	256	352	634			

Table 2.5: Multiple bus message transfer latencies (μsec). All values are ± 1

	Buses		
ρ	3	4	5
0.20	152	118	112
0.25	251	133	118
0.30		165	127
0.40			176

The results for the preemptive circuit-switched and virtual cut-through message transfers can be compared on the basis of both the overall average delay and the delay experienced by each message distance. The average delays for $\rho = 0.2 \ldots 0.5$ are shown in figure 2.7. Here, for low ρ, the average latencies for each γ value are very similar. However, as the utilization is increased, the virtual cut-through message transfers incur less delays. This is due to the increasing effects of overheads on the preemptive circuit-switched transfers. As loads increase, more preemptions occur, each preemption adding a fixed delay to the message transfer. For virtual cut-through transfers, no extra overhead is added when the message is stopped, other than a single-unit pass-through delay, which is limited to the number of intermediate gates traversed by the message. The amount of overhead for preemptive circuit-switched transfers is not limited; a message may be preempted many times and will incur the same overhead for each preemption.

The results for the individual distances (Figure 2.8-2.11), show that for $H < 3$, the virtual cut-through and preemptive circuit-switched message transfers are very similar. For long-distance message transfers, the preemptive circuit-switched latencies are strictly decreasing while the virtual cut-through results still exhibit an increasing trend, mainly due to the added overhead for each gate traversal. Since a similar over-

head is included in the preemptive circuit-switched transfers, for very localized loads, the preemptive circuit-switched technique is the method of choice. In fact, similar methods have been used in previous architectures to some effect. The Homogeneous Processor architecture [25] has used a bus connection mechanism which in effect sets up a circuit-switched path to the neighboring processor's memory. We are unable to compare results with this architecture since message transfers for longer distances are handled by a separate mechanism.

For less localized message transfers, ($\gamma = 0.3$), the difference between virtual cut-through and preemptive circuit-switched still exists. However, in these cases, the lower distance transfer latencies are adversely affected in the preemptive circuit-switched technique. We attribute this to the priority selection rules for the preemptive circuit-switched transfers. Since the priority selection does not incorporate γ, as the volume of longer distance messages is increased, the added effects are not compensated for.

Comparing the results for increasing system utilizations, it can be seen that the preemptive circuit-switched technique is less sensitive to increases in the load. The highest average latency for ρ=0.5 does not exceed 400 μsec for all γ values examined. As the system load is increased, the effect of the load is distributed to all message distances resulting in a series of shifted graphs. The virtual cut-through results indicated that the technique is very sensitive to load changes. Longer distance transfers are more severely affected with increased loads in the virtual cut-through case.

Interpreting the characteristics of the two transfer methods at differing loads, it can be seen that both methods offer distinct advantages. If a bound on the maxi-

mum latency is desired and local messages can incur higher delays, the preemptive circuit-switched method can be used. If on the other hand, local message transfers are most important, and longer distance transfer delays do not severely affect the system behavior, then the virtual cut-through technique should be used since the local latencies are significantly lower than the latencies in the preemptive circuit-switched case.

The utilization results (see table 2.6 for some samples), match the predicted values found in the load analysis. This confirms the validity of the load analysis and the comparisons with the multiple bus in this respect.

Table 2.6: Comparison of simulation and analytical method effective (ρ) confidence intervals for each segment. γ=0.3, N= 10, λ=25000 messages/sec, μ = 10000 messages/sec/bus

	Segments									
Method	1	2	3	4	5	6	7	8	9	10
Analytical ρ	0.24	0.31	0.32	0.32	0.31	0.31	0.32	0.32	0.31	0.24
PCS $\rho_{simul}(\pm 0.01)$	0.23	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.31	0.23
VCT $\rho_{simul}(\pm 0.01)$	0.23	0.32	0.31	0.32	0.31	0.31	0.32	0.31	0.31	0.23

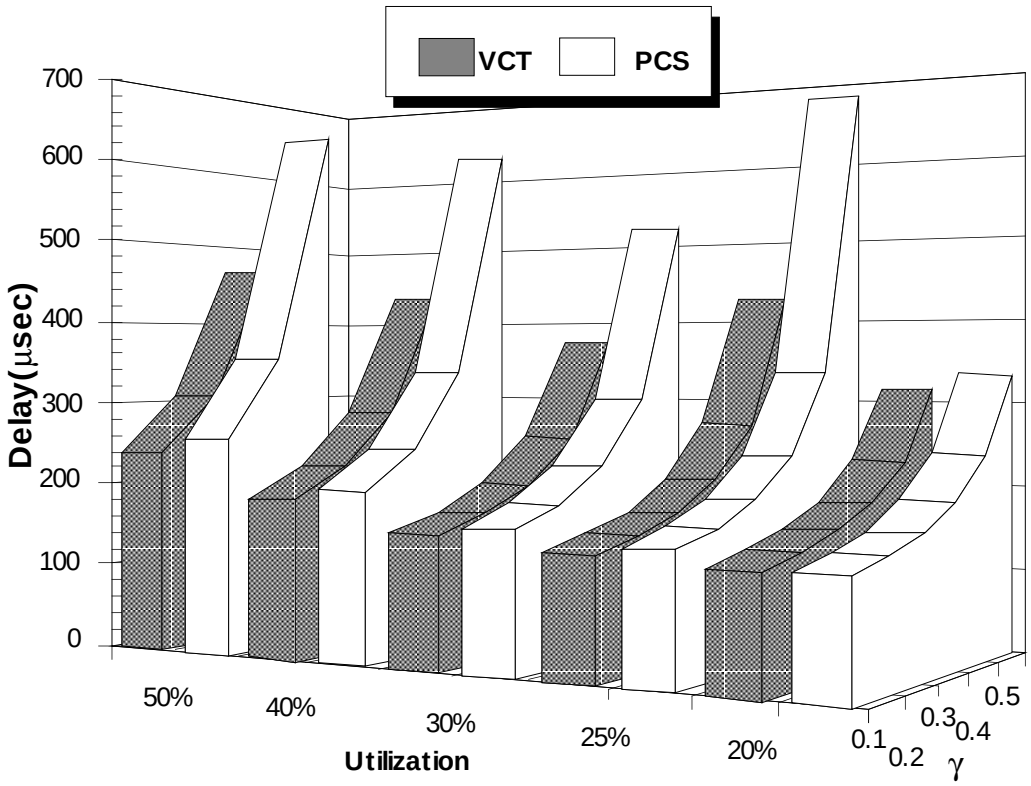


Figure 2.7: Average latencies (μsec) vs γ for virtual cut-through and preemptive circuit-switched message transfers with ρ=0.2 ... 0.5

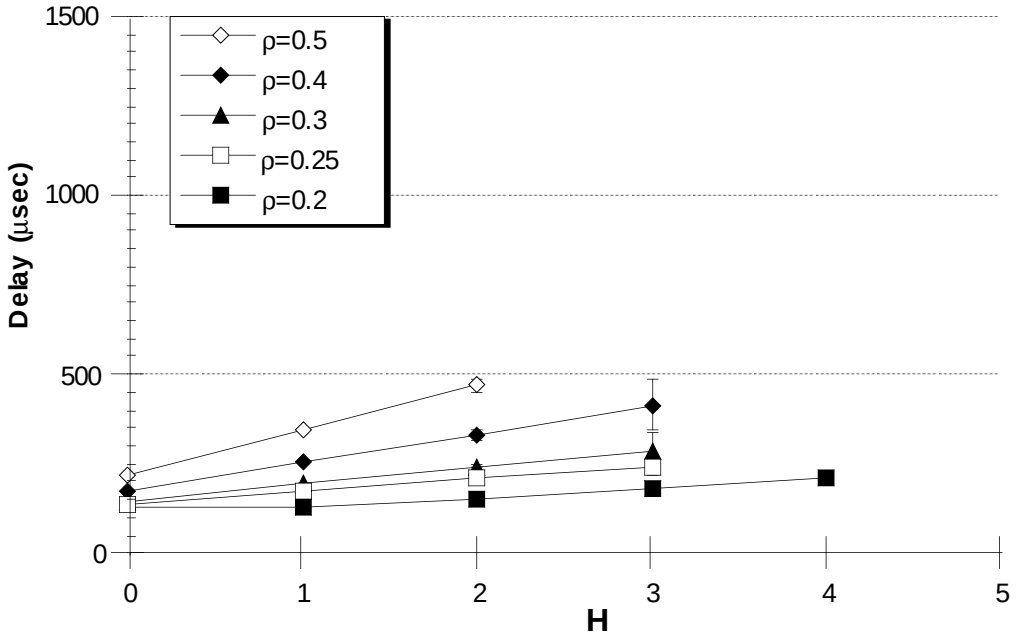


Figure 2.8: Virtual cut-through message transfer delays for $\gamma = 0.1$ with ρ=0.2 ... 0.5

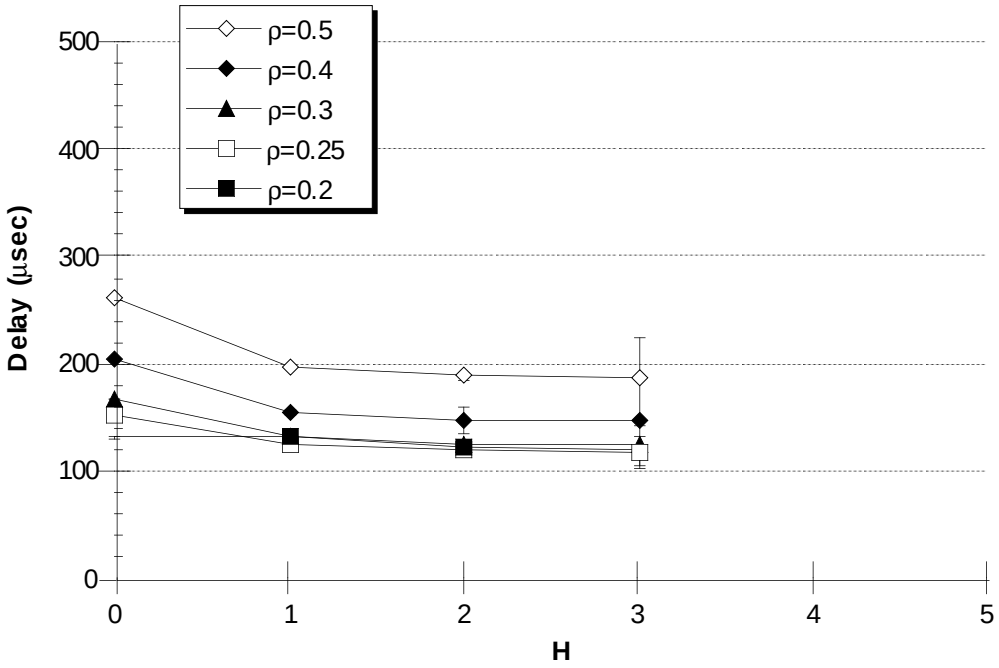


Figure 2.9: Preemptive circuit-switched message transfer delays for $\gamma = 0.1$ with ρ=0.2 ...0.5

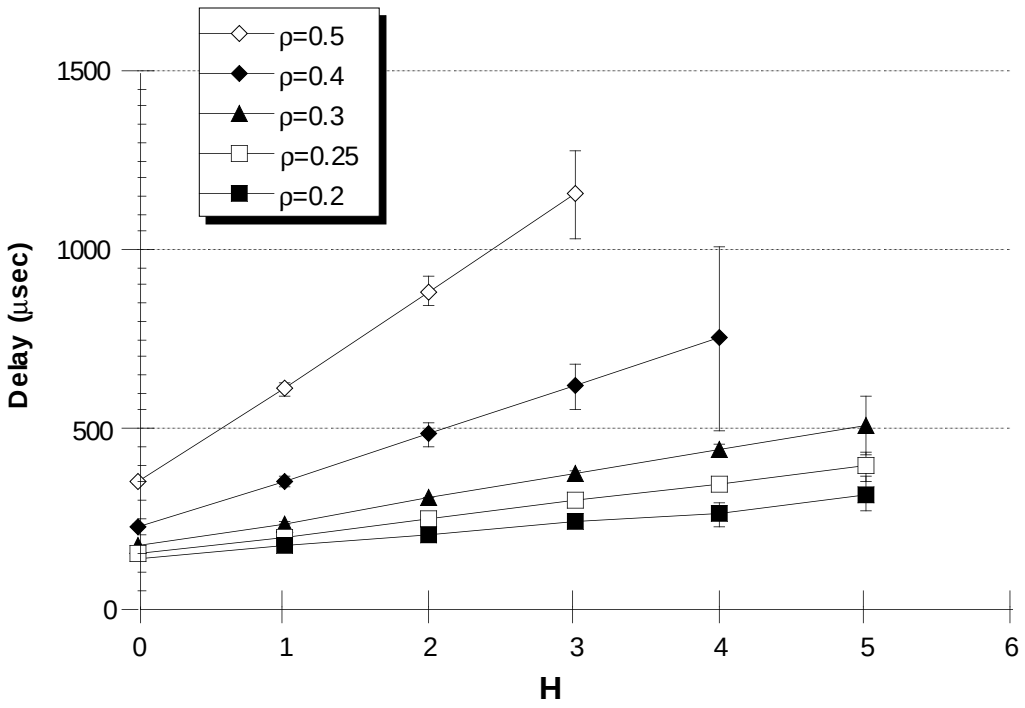


Figure 2.10: Virtual cut-through message transfer delays for $\gamma = 0.3$ with ρ=0.2 ...0.5

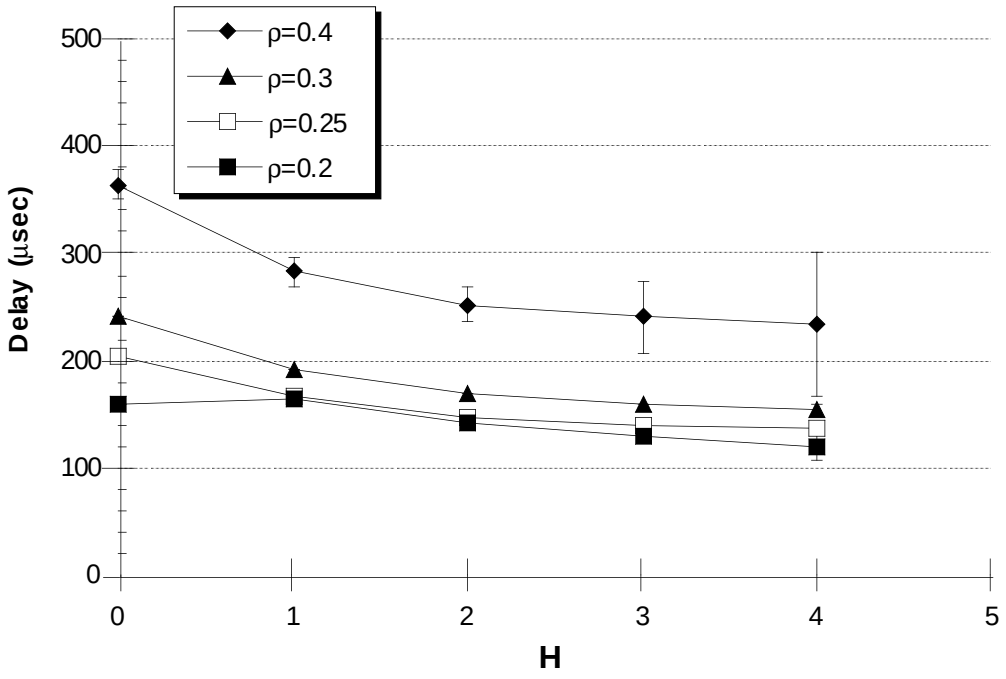


Figure 2.11: Preemptive circuit-switched message transfer delays for $\gamma = 0.3$ with ρ=0.2 ...0.4

2.5 Implementation of the Segmented Bus

The segmented bus hardware needs to satisfy two objectives: (i) dynamic creation, removal and resizing of segments, and (ii) data transfers within and across segments. We propose to achieve these objectives using a building block consisting of a buscon, a device to form buses, and a gate, to transfer data across segments. An additional unit called the arbus is proposed for priority arbitration over multiple segments. The basic hardware for the segmented bus is shown in figure 2.12. In the figure, three buscons make a segment. Two buscons at the ends are disconnected to form the segment boundaries. At the ends of the segment, two gates transfer data to adjacent segments.

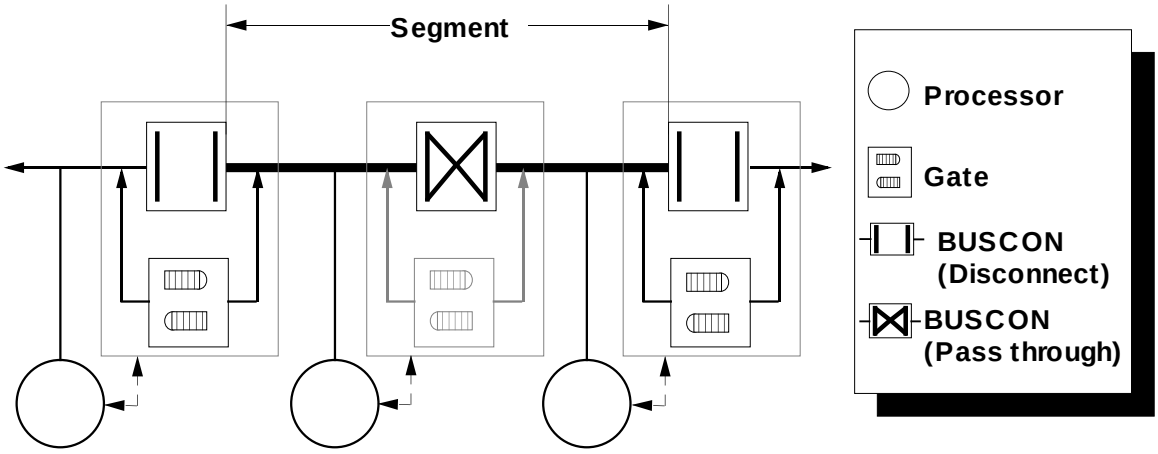


Figure 2.12: Segmented bus hardware

2.5.1 Data Transfer Operations on the Segmented Bus

Intra-segment message transfers use the segment. Since only message passing is needed, for simplified addressing, a multiplexed address and data bus will be used. During a message transfer m_i, after bus arbitration and gaining of bus mastership, $P_{r(i)}$ is placed on the bus. After this value, the message transfer is started. Message transfers end when the processor asserts an *end-of-message* signal on the bus. On the receiving end, a FIFO buffer will be used so that all incoming messages can be queued preventing the overwriting of consecutive messages.

Inter-segment message transfers build upon the basic protocols for intra-segment transfers. A processor does not distinguish between intra and inter-segment transfers. If $P_{r(i)} \notin \Pi^j$, gates at either end of s^j compare the value to the bounding values Π^j_L and Π^j_H. If the limits are exceeded, then one of the gates will start an inter-segment transfer using the preemptive circuit-switched or virtual cut-through message transfer technique.

2.5.2 Buscon

The buscon device is a switch to connect and disconnected bus lines. A segment is created by attaching multiple buscon devices to form a bus. This bus is electrically similar to a standard bus such as the VME [44], however, there are fewer signals since this is only a message passing bus. Any processor residing on the segment can send a message to another processor on the same segment without passing through a gate.

A key assumption in this work is that bus segments can be dynamically joined to-

gether to form longer single buses. This must be done in such a way that the electrical characteristics of the longer bus are still satisfactory, i.e., short propagation delays, proper termination, low noise, etc. There are essentially three methods to isolate and join signal lines: CMOS switches, bipolar tristate buffers and electromechanical relays. CMOS switches based on transmission gates [45] have been used in several on-chip switching networks such as the Gated Connection Network [30]. However, several series connections, even with pre-charge circuitry, significantly increases the delay of transmission. Bipolar buffers have small delays (≤ 3 ns) and the total delay with up to 10 processors per segment does not exceed 30 ns. This is still a significant latency when compared to the minimum cycle time of the VME Bus which is about 100 ns [46]. In addition, since bipolar switches are unidirectional, extra hardware is necessary to change the direction of each buffer depending on the location of the buffer with respect to the source and destination of the message. This can be overcome using two buses, one for each direction, at an increased cost.

We propose the use of microrelays to connect and disconnect bus lines. Commercial low level signal relays have 1 ms switching times and can switch signals of up to 50 Mhz. The major advantages of relay -based switching compared to other electronic switching methods are the low contact resistance ($R_c < 0.1\Omega$), allowing long chains of switches, and the low capacitance of the switch, reducing effects of loading. Although relays are a primitive form of switching, all other electronic means suffer from linearly increasing delays which degrade the broadcast capability of the bus.

Since the purpose of reconfiguring the segmented bus is to optimize the architecture to changing computation demands of the processes, the target application

area determines the rate of reconfiguration. For autonomous systems such as mobile robots, the cycle time (the fastest changing values in the system) is in the order of 10 ms [47]. With this in mind, major changes in the process computation demands and task communication graph are not expected to occur within several orders of magnitude of this value, resulting in 10-100 minute reconfiguration intervals. This long interval allows the use of slower switching devices for bus reconfiguration.

It must be noted that the buscon device does take time to reconfigure the bus connections; this is one disadvantage of the electromechanical solution. If the system is to be operated in environments where such delays are unacceptable, the bipolar buffer solution is still available at a higher cost.

2.5.3 Gate

The gate is a device used to transfer inter-segment messages. The hardware and operation of the gate for both message transfer techniques are described in this section.

Gate Hardware

The hardware for a gate is similar for all transfer techniques (Figure 2.13). It consists of a buffer used to store messages, inter-segment message detection and arbitration modules.

Buffer: There is a message buffer in all versions of the gate. The buffer provides temporary storage for inter-segment messages, blocked on a segment.

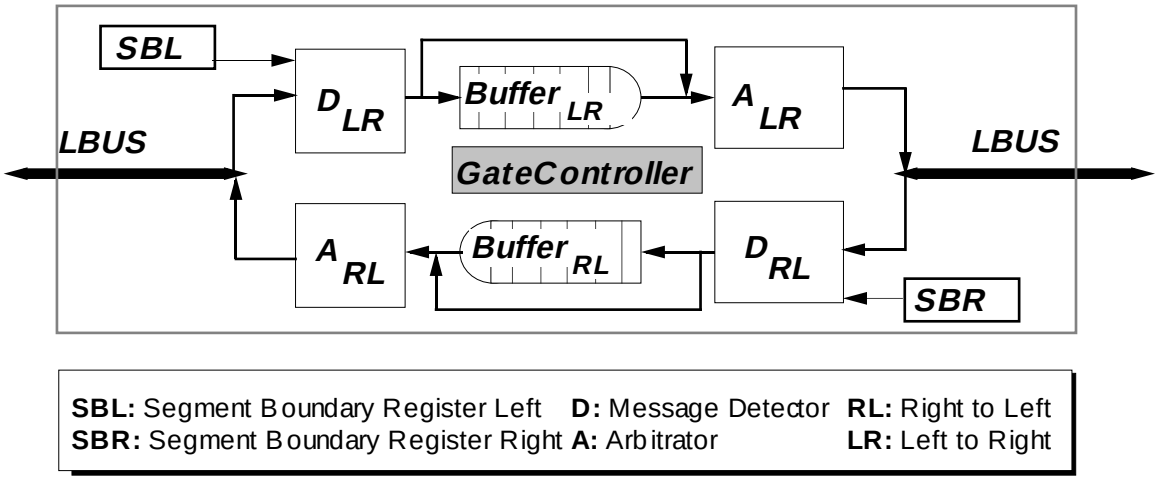


Figure 2.13: Gate hardware organization

The buffers operate in a FIFO fashion. Since there are two directions of data flow, each gate has two buffers for *left-to-right* and *right-to-left* message transfers. These buffers operate independently.

Message Detector: The primary function of the gate is the transfer of inter-segment messages. The message detector detects messages destined to processors not local to the segment. Two registers named *Segment Boundary Left* (SBL) and *Segment Boundary Right* (SBR) keep the Π_L^j and Π_H^j. All message destination addresses are compared to these registers. If an inter-segment message is detected, the buffers are enabled and the gate controller initiates an inter-segment transfer. Since each gate is attached to a processor, the values in the registers can be changed via software for system reconfiguration.

Arbitrator: Each message transfer requires the determination of priorities for sequencing of messages within the gate. The arbitrator compares π_i with the

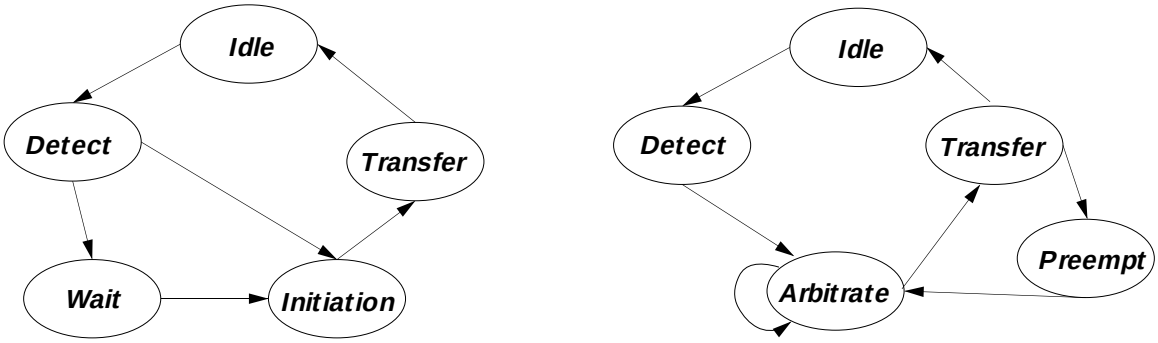
other active message transfers, and initiates inter-segment message transfers on the new segments. In a message transfer, the arbitrator functions are highly dependent on the type of message transfer. For virtual cut-through transfers, the arbitration mechanism is local and only arbitration on the next segment is required. For preemptive circuit-switched transfers, the arbitrator may act either as an *originator* or as a *passer*. As an originator, the arbitrator is required to setup the path of a message. The arbitrator decides to start a new transfer on the next segment based on the priorities of the message. An arbitrator may also act as a passer, where it sets up the internal data path of the gate to form a circuit-switched path for the message. In this case, it responds to the signals from other arbitrators to preempt ongoing transfers.

Gate Operation

The operation of the gate for all data transfers starts with the *idle* phase where the signals on the segment are monitored. Upon the start of bus activity, the *detection* phase is selected and the $P_{r(i)}$ of a new message is detected. For example, on s^j if the message is decoded by the left gate and the $P_{r(i)} < \Pi_L^j$ then the message is accepted into the gate. If the message is decoded on the right side, then $P_{r(i)} > \Pi_H^j$ is checked to decide whether the message should be accepted.

After the detection phase, the *initiation* phase is entered. The operation of this and following phases is dependent on the data transfer technique.

For virtual cut-through message transfers (Figure 2.14a) , if the next segment is free, the *wait* state is skipped and a new transfer is immediately started from the



a) VCT Task State Diagram **b) PCS Task State Diagram**

Figure 2.14: Virtual cut-through and preemptive circuit-switched message transfer state diagrams

initiation phase. The *wait* state is entered only if the new transfer fails. If the wait state is entered, the transition to the initiation phase occurs when the next segment becomes free and the bus is granted to the gate.

Preemptive circuit-switchedmessage transfers (Figure 2.14b) also start with the detection phase. After the inter-segment message transfer is detected, the *arbitrate* phase is entered. Since circuit-switched message transfers require the allocation of the whole path, the arbus is used to arbitrate for all the required segments. If the path is available, the *transfer* phase is entered, otherwise, the data is transferred into the gate to wait for the availability of the path.

In the *arbitrate* state, the gate tries to set up the path by retrying after the completion of each transfer on the requested segments is completed. When the path becomes available, the transfer phase is entered and the message transfer is started. During this phase, the message transfer may be preempted. If this occurs, the path

needs to be setup again to resume the message transfer when the priority of the requested segments is less than the priority of the preempted message.

2.5.4 Arbus

The arbus is a bus used for the arbitration of inter-segment path set up requests during preemptive circuit-switched message transfers. For P processors and L priority levels, the arbus requires $\log P + \log L + 2$ signal lines. Like segments, the arbus is also formed by the connection of buscon devices. However, the arbus only passes information between the two gates at the ends of each segment (inactive segments simply pass-through the signals). When a message transfer $m_i = (\tau, \{j \ldots k\}, \pi_i)$ is started, the following sequence of events occur for the arbitration of multiple segments:

1. The originating processor $P_{o(i)}$ first gains hold of s^j. For this to occur, $\pi_i > \pi_{\Omega^j}$ must be satisfied.

2. The message header is read by the appropriate gate.

3. The gate decoding the header places the values of s^j and π_{new} on the arbus.

4. The information placed on the arbus is decoded by the gate between s^j and s^{j+1}. If s^{j+1} is free or if $\pi_i > \pi_{\Omega^{j+1}}$ then the message Ω^{j+1} on the next segment is preempted. If $j + 1 \neq k$ the information on the arbus is passed on to the next gate. if $\pi_i \leq \pi_{\Omega^{j+1}}$ then a signal is sent back to inform the originating gate about the unavailability of the path.

5. The decode-preempt cycle continues until either a segment transferring a high priority message is found or s^k is reached.

Preemptive circuit-switched message transfers require the arbus to set up paths, and additional lines for preemption. The gains for the additional hardware for the preemptive circuit-switched transfer technique are both in the reduction of latencies and the added capability to transfer messages with real-time deadlines which has been examined recently [48, 49].

2.6 Discussion

As multiprocessors are increasingly applied to the control of autonomous systems requiring extensive computational power, the interprocessor communication network will become the bottleneck of the computer architecture. Exploiting the characteristics of the processes to be executed and integrating this onto the architecture offers a method for expandability and increased performance.

The segmented bus is an effective interconnection network for multi-processors with localized data. Virtual cut-through and preemptive circuit-switched data transfers take advantage of the multiple segment communication channel and provide fast data transfers. Depending on the distribution of loads, either technique may be used to minimize the transfer time of a selected message distance. For highly localized data, preemptive circuit-switched message transfers provide low latencies not only for inter-segment messages but also for local data. For less localized cases, depending on the outgoing message factor, preemptive circuit-switched or virtual cut-through techniques may be employed; virtual cut-through transfers are effective for most localization factors but have strictly increasing latencies as the message distances increase.

The hardware implementation of preemptive transfers is expected to be more complex than any of the other techniques except the multiple bus. However, it must be noted that the preemptive bus priorities can be changed to allow deadline based message scheduling, a method that will be required if real-time tasks will be executing on the system. In this case, the predictability and low bounds of the preemptive technique justify the more complex hardware needed to implement the preemption operation.

One observation that can be made from the results is that the utilization of the middle segments on the segmented bus is typically likely to be higher than the side segments. This was clearly observed in our simulations and analytical evaluations.

Chapter 3

Real-Time Communication Networks

A *real-time message stream* M_i is a unidirectional flow of data within a packet-switched network with a deadline for the end-to-end delivery of each message. A *message* $m_i \in M_i$ is one of the messages in message stream M_i. The types of real-time applications addressed in this work are all periodic. As a result, a message stream represents a periodic transmission of a message from one source to one destination.

The temporal properties of a real-time message stream M_i are defined by the transmission time per message[1] τ_i and the period T_i of the message stream. If the message stream source is not purely periodic, the linear bounded arrival process model [50] can also be used to define the minimum interarrival time which can still be expressed as a periodic stream. The service requester specifies two other parameters which define the acceptability of the QOS provided by the network. These are the message maximum end-to-end latency D_i, and the jitter or variance of this latency,

[1] We assume all link bandwidths are equal, which results in a constant τ for each message, regardless of which link it is transmitted on.

J_i. The network-offered maximum end–to–end latency for this stream is denoted by D'_i; $D'_i \leq D_i$ for a stream in the network. The difference $S_i = D_i - D'_i$ is called the *end-to-end slack* of a message stream.

Real-time message streams are transported over simplex links in a network which consists of nodes connected by links. Γ_i denotes the set of links forming the path traversed by the stream. $H_i = |\Gamma_i|$ denotes the length of the path. Conversely, the streams passing through a link j are denoted by Ω^j. At each node traversed, a switch is used to select the next link on the path of the message. Table 3.1 summarizes the notation, and Figure 3.1 illustrates the use of the notation with an example. In this example, there are two message streams: M_1, following path $\Gamma_1 = \{AC, CD, DE\}$; and M_2, following path $\Gamma_2 = \{BC, CD, DF\}$. For link CD, $\Omega^{CD} = \{M_1, M_2\}$.

Table 3.1: Notation for real-time message streams

M_i	Message stream i
m_i	A message in stream M_i
ℓ^j	Link j
T_i	Period
τ_i	Transmission time
D_i	Maximum source-to-destination delay
D'_i	Network offered maximum source to destination delay
S_i	End–to–end slack
J_i	End–to–end delay jitter
Γ_i	Message path (set of links)
H_i	Number of hops on path ($\|\Gamma_i\|$)
Ω^j	Set of streams served by ℓ^j

The provision of real-time services within a packet-switched network has recently received increased attention. Methods that offer real-time [21, 51] or both real-time

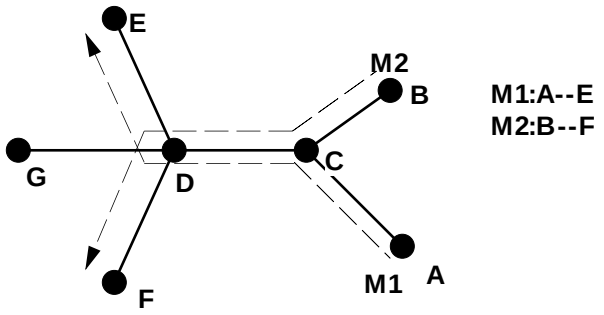


Figure 3.1: Example network

Table 3.2: Real-time properties of previous methods

Method	End–to–End Delay Bound	Jitter	Data Loss	RT Classes	Non-RT Classes
EDD-K[21]	$[H\tau, HT]$	high	none	unlimited	No
EDD-DASH[52]	$[H\tau, HT]$	high	none	unlimited	Yes
EDD-S[2]	$[H\tau, HT]$	high	none	limited	Yes
EDD-SJ[53]	$[H\tau, HT]$	low	none	limited	Yes
Stop-and-Go[51]	$[HT, 2HT]$	low	none	limited	No
Framing[54]	$[H\tau, HT]$	high	possible	1	Yes
MARS[55,56]	$[H\tau, HT]$	high	possible	1	Yes
WFQ[17]	$[HT, 2HT]$	high	none	limited	Yes

and statistical [2, 17, 52–56] QOS guarantees are available. Table 3.2 summarizes the delay and loss QOS provisions of these techniques. In the table, H denotes the number of hops or links traversed by the message, T denotes the period (or minimum interarrival time) of the real-time message stream, and τ denotes the time to transfer the message on a link, excluding processing, queueing and propagation delays. Note that in all proposed real-time message transfer schemes, a lower limit of $H\tau$ exists for the guaranteed end-to-end delay of a message.

For interactive or critical real-time applications such as video-conferencing or com-

mand/control systems [16], the end-to-end delay of a message must not only be guaranteed, but must also be very low. Since existing methods have a lower bound of $H\tau$ for end-to-end delay, as shown in Table 3.2, they may not be suitable for such applications. A circuit-switched method, in contrast, has a lower limit on end-to-end-delay of τ.

During real-time message transmission, a mechanism to enforce end–to–end delay bounds is needed. The end–to–end delay can be enforced by bounding individual link delays. This requires guaranteed access to the link at predetermined intervals. Real-time schedulers provide guaranteed access to a shared resource; they have been successfully used to schedule processes on a central processing unit. Several types of schedulers exist: Rate Monotonic [57], Earliest Deadline First [57] and Least Slack [58] (For a detailed review, see [59]). Real-time schedulers use priorities and preemption to guarantee access to resources. Priorities determine the user of the resource. Some schedulers, such as Rate Monotonic, assign a fixed priority to each resource user. Others, such as Earliest Deadline First, dynamically change priorities. For Earliest Deadline First, at any time, the user with the earliest deadline has the highest priority. When the priority of the current user is lower than the priority of a waiting user, it is preempted to let the waiting user access the resource. This assures that users with high priorities, whether dynamically or statically assigned, use the resource first. The preempted user will regain the resource at a future time, with no loss of data.

3.1 Low Latency Message Transmission

Considering a link as a shared resource, any real-time scheduler can be used to regulate the usage of a communication link. For real-time message streams, the Earliest Deadline First (EDF) scheduler has been preferred [2, 17, 21, 22], since it can prioritize access to a link based on delay bounds. To certify that the all delay bounds are met, the schedulability of the message streams on the link must be assured, as defined below:

Definition 3.1 (Link Schedulability) *Message Streams $M_i \in \Omega^j$ are* schedulable *using an Earliest Deadline First scheduler if and only if it is certain that all $m_i \in M_i$ will be fully transmitted before their deadline.*

Real-time schedulers have been designed for a process model. A process is ready for processing at a certain time instant and it needs to be completed before its delay bound. With EDF, the *ready time* r and the *delay bound* d of a process are used to compute the deadline $z = r + d$ by which the process execution must be completed. The process with the earliest deadline z is selected for execution. The total time a process waits for execution cannot exceed $w = d - \tau$. The relationship of these parameters is shown in Figure 3.2, and they are summarized in Table 3.3. In process scheduling, since the regular instantiation of the process is assured, the deadlines are set to be $z = kT + d$ by using the multiples of the period T of the process as a reference point. Since a process must be completed before a new instantiation, $d \leq T$.

In the communications domain, the process model has no exact equivalent. On communication links, a message m_i arrives to a node *over time*. The header time

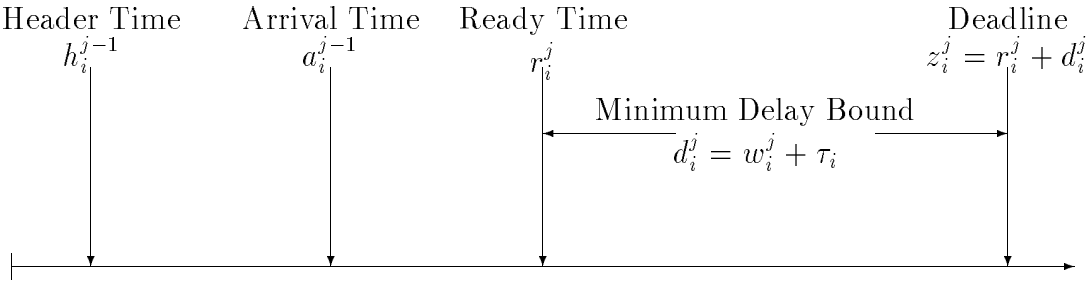


Figure 3.2: Deadline-based scheduler parameters

h_i^{j-1} denotes the time when the first decodable part of a message (typically header of length τ_h) is received on ℓ^{j-1} and arrival time a_i^{j-1} denotes the time the whole message arrives to the node from ℓ^{j-1}. The terms used are summarized in table 3.3. The link indices are relative to a node, with the ℓ^{j-1} denoting the *incoming* and the ℓ^j denoting the outgoing link. For example, suppose a message is sent from node C to node F in figure 3.1. At an intermediate node D, ℓ^{j-1} would be link CD, and ℓ^j would be link DF. We also note the relationship on a link $h_i^{j-1} + (\tau_i - \tau_h) \leq a_i^{j-1} = z_i^{j-1}$ (see Figure 3.2). The deadline z_i^j bounding the transmission of m_i on link j is set relative to its ready time, r_i^j. Therefore the instant where the message is considered to be ready is important. In [2,21], r_i^j is set to a_i^{j-1} or the next interarrival time, whichever is later. In [51,53], r_i^j is set at an even later point for jitter reduction.

Table 3.3: Notation for link operation parameters

h_i^j	Time header of m_i arrives to node via ℓ^j
a_i^j	Time transmission of m_i on ℓ^j is completed
r_i^j	Time a message is ready to be sent on ℓ^j
d_i^j	Delay bound for m_i on ℓ^j
z_i^j	Message deadline $z_i^j = r_i^j + d_i^j$ on ℓ^j
w_i^j	Link waiting time $w_i^j = d_i^j - \tau_i^j$ on ℓ^j

If $r_i^j \geq a_i^{j-1}$, this means the message is considered ready only after it is received in whole. This is the same as the store-and-forward mechanism used in packet-switched transmission. If the message is transmitted in τ time units over a single link, the minimum end–to–end delay for this message will be $H\tau$. To obtain performance closer to that of circuit-switching, message transmissions need to be overlapped. By this we mean that different portions of a single message must be transmitted at the same time on multiple links along its path. This requires the message service on a link to start before the transmission ends on the previous link; i.e. $r_i^j < a_i^{j-1}$.

The idea of overlapping packet transmissions, called *virtual cut-through*, was first proposed by Kermani and Kleinrock [40]. This work was defined for networks using statistical multiplexing, and did not address the needs of real-time messages. The first problem addressed is how to use cut-through for real-time message streams. Using cut-through, guaranteed end–to–end delays can be reduced to τ, which is the same delay seen in circuit-switching. By reducing the minimum delay of real-time messages, the network can serve applications which otherwise could not be accepted. A clearer statement of the problem is the following:

Problem 1: Given a message stream M_i with parameters (τ_i, T_i, D_i, J_i) and path Γ_i, find a message transfer technique which can offer a lower bound τ on the end–to–end delay bound D.

3.2 Implementation of Real-time Message Services

A goal of this research is to develop effective methods of real-time message transmission in B-ISDN networks. Because ATM is the transport method for B-ISDN, the implementation issues of using ATM need to be addressed. As explained earlier, the processing time per cell for ATM must be kept under one microsecond. The Qualities of Service that are required for real-time communication, however, demand a significant amount of processing by the nodes (or switches) along a path. This overhead is due to the scheduling of individual cells at each node.

An analogy can be made with routing of messages. Routing of messages also requires far more time than 1 microsecond to perform. Instead of routing each individual cell, routing is accomplished during channel establishment between the source and destination nodes. Thereafter, the one-time cost for routing is shared by all of the cells on that channel. This effectively results in a low overhead cost per cell. Similarly, deadline-based message scheduling needs to be performed above the cell level.

Problem 2: Given a message stream M_i, whose messages are segmented into cells $c_{i,1}, c_{i,2}, \ldots, c_{i,k}$, find a transport method which provides real-time message guarantees, but using the QOS mechanisms that have been proposed for ATM to the greatest degree possible. The overhead per cell required for message scheduling must be only a fraction of the total time available for processing each cell.

3.3 High Availability Network Resource Allocation

Integrated networks are connection oriented. During a connection request, the network computes the resources required to satisfy the requested QOS level and allocates these resources to the stream for the duration of the connection. For real-time message streams, the QOS check involves the verification of the real-time scheduler's ability to transmit a message within its end–to–end delay bound D.

In the verification procedure, the schedulability of the message stream at each link on its path is computed first. After the schedulability check, the hard end-to-end delay bound of the message is computed. Both computations require a parameter that is unavailable. In order to verify the schedulability of ℓ^j, temporal parameters T_i, τ_i and d_i^j must be known for each stream $M_i \in \Omega^j$. However, for real-time message streams, only an end-to-end delay bound is specified. The individual links may complete the delivery of the message in an arbitrary length of time as long as the end–to–end requirement is met. In order to compute the minimum end–to–end latency, again, the minimum delay bound $d_i^j \ \forall \ell^j \in \Gamma_i$ is necessary. One solution [2,21,22] to this predicament assumes that message streams are requested one at a time. In this case, the parameters for the existing set of messages Ω^j have already been computed. We find the minimum delay bound $d_i'^j$ for the new message stream M_i that will preserve the schedulability of the augmented message set $\{\Omega^j \cup M_i\}$.

Once the minimum delay bounds are computed, the end–to–end delay can be determined. The end–to–end delay in the network will be no greater than network-

offered end–to–end delay bound D'. This value may be less or greater than the requested delay D. With this procedure, a real-time message stream M_i may be rejected if: (i) M_i is not schedulable on some $\ell^j \in \Gamma_i$, or (ii) $D'_i > D_i$.

Current methods do not have any fallback solutions when a message admission request is denied due to the failure of either condition. The problem addressed in this dissertation deals with increasing the likelihood of message stream acceptance. We use the fact that the sender and the receiver of a message stream are insensitive to the latencies at each intermediate link as long as the end-to-end latency guarantees are maintained.

Problem 3: Given is a network and a set of real-time message streams which have already been admitted to this network. Also given is a new message stream M_i, which cannot be admitted due to a scheduling failure on one or more links. Find a dynamic method for altering the delay bounds of messages previously scheduled on those links, so that (i) M_i can be successfully scheduled $\forall \ell^j \in \Gamma_i$, and (ii) $D'_i \leq D_i$.

Chapter 4

Low Latency, Low Jitter Message Transmission

Current communication networks carry real-time messages streams over dedicated channels. Emerging integrated networks intend to merge real-time and delay insensitive data streams into a single network. Upon this merger, the Qualities of Service enjoyed in dedicated channels must be provided in the integrated network as well.

This chapter presents a new real-time message transfer method. The method, called *preemptive cut-through*, is intended for the transmission of critical real-time streams over integrated networks. These streams require low latency, low delay-jitter and zero data loss. First the theoretical foundations of preemptive cut-through are defined through the concept of *arrival over time*. The reduction in end–to–end delay is proven theoretically. Preemptive cut-through relies on time-stamping of messages. A new relative time-stamping technique termed δ*-stamping* is described.

Section 4.4 considers the implementation of preemptive cut-through on an ATM [15]

network. A *message-level* delay control mechanism for ATM is described. The chapter concludes with a discussion of preemptive cut-through and a comparison with related work.

4.1 Arrival Over Time

On communication links, a message m_i arrives to a node *over time*. Let h_i^{j-1} denote the time at which the header of m_i is sent over ℓ^{j-1}. With the ready time at r_i^{j-1}, the header time must satisfy $r_i^{j-1} \leq h_i^{j-1} \leq r_i^{j-1} + \kappa_i^{j-1}$ to transmit m_i within its delay bound.

When the header of m_i arrives at the next node, the ready time on the next link ℓ^j must be computed. In previous works [2,21,53], $r_i^j \geq r_i^{j-1} + d_i^{j-1}$ or even $r_i^j \geq r_i^{j-1} + T_i$ [51]. This means, the message is received in its entirety before it is considered ready for transmission. Although this restriction is sufficient to guarantee a bound on the end-to-end delay, it is not necessary. With $r_i^j \geq r_i^{j-1} + d_i^{j-1}$ and $d = \tau$ for all $\ell \in \Gamma_i$, the minimum end-to-end delay will be $H\tau$.

To obtain performance closer to that of circuit-switching, r_i^j must be set to an earlier time while still assuring that the delay bound is never violated. In the following, we prove the necessary conditions for the value of r_i^j, given the ready time on the previous link, r_i^{j-1}.

Lemma 4.1 (Deadlines) *If m_i is transmitted on ℓ^{j-1} and is being considered for transmission on ℓ^j, $z_i^j \geq z_i^{j-1}$.*

Proof: By contradiction. Assume M_i is schedulable and set z_i^j such that $z_i^j < z_i^{j-1}$. In the worst case, the transmission of m_i on ℓ^{j-1} can be started at $h_i^{j-1} = z_i^{j-1} - \tau$. This means, the last bit of m_i will be transmitted at z_i^{j-1}. Since $z_i^{j-1} > z_i^j$, all data arriving in $[z_i^j, z_i^{j-1}]$ will have missed their deadline on ℓ^j, violating the schedulability assumption. ■

Lemma 4.2 *If m_i is transmitted on ℓ^{j-1} and is being considered for transmission on ℓ^j, m_i will be transmitted within its delay bound d_i^j if and only if $r_i^j \geq \max(h_i^{j-1}, r_i^{j-1} + d_i^{j-1} - d_i^j)$.*

Proof: The first condition, $r_i^j \geq h_i^{j-1}$, must hold at all times since r_i^j can only be set when the header of m_i arrives.

For the second condition, from lemma 4.1, $z_i^j \geq z_i^{j-1}$. Using the definition of $z_i^j = r_i^j + d_i^j$,

$$r_i^j + d_i^j \geq r_i^{j-1} + d_i^{j-1} \tag{4.1}$$

$$r_i^j \geq r_i^{j-1} + d_i^{j-1} - d_i^j \tag{4.2}$$

Combining the two, we obtain $r_i^j \geq \max(h_i^{j-1}, r_i^{j-1} + d_i^{j-1} - d_i^j)$. ■

Theorem 4.1 (Worst case Ready Time) *If m_i is transmitted on ℓ^{j-1} and is being considered for transmission on ℓ^j, in the worst case, $r_i^j \geq r_i^{j-1} + \kappa_i^{j-1}$.*

Proof: From lemma 4.2, $r_i^j \geq \max(h_i^{j-1}, r_i^{j-1} + d_i^{j-1} - d_i^j)$. The maximum value of the header transmission time h_i^{j-1} is $r_i^{j-1} + \kappa_i^{j-1}$. Since $r_i^{j-1} + d_i^{j-1} - d_i^j = r_i^{j-1} + \kappa_i^{j-1} - \kappa_i^j$,

$r_i^{j-1} + \kappa_i^{j-1}$ is always greater than or equal to $r_i^{j-1} + d_i^{j-1} - d_i^j$. Thus, in the worst case, $r_i^j \geq r_i^{j-1} + \kappa_i^{j-1}$. ■

Since critical applications must consider the worst case delay, theorem 4.1 must be used to compute the end-to-end delay bound.

4.2 Preemptive Cut-Through

From theorem 4.1, it can be seen that r_i^j assigned to m_i need not be at $r_i^{j-1} + d_i^{j-1}$ as in store-and-forward. When the earlier ready time is exploited, message overlap will occur. Figure 4.1 illustrates this. The total end-to-end latency seen by message m_i is the following:

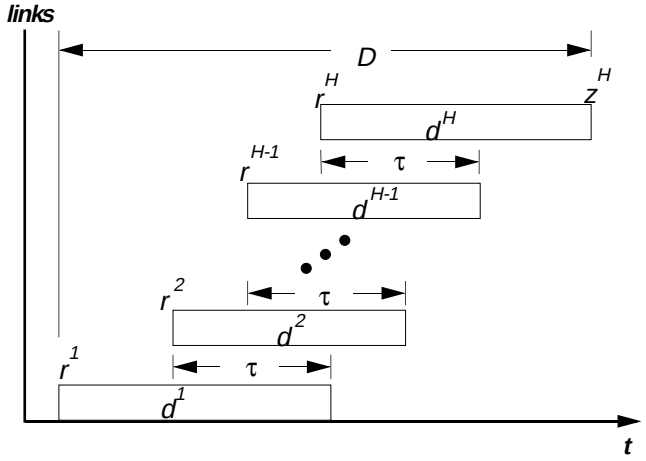


Figure 4.1: Total end-to-end latency

$$D = (r_i^2 - r_i^1) + (r_i^3 - r_i^2) + \cdots + (r_i^{H-1} - r_i^{H-2}) + (r_i^H - r_i^{H-1}) + d^H \qquad (4.3)$$

Consider the case in which r_i^j is set equal to $r_i^{j-1} + \kappa_i^{j-1}$, for all j. This is the case when the ready time is set as early as possible, without violating theorem 4.1. Substituting into (4.3) and converting to a summation, we obtain,

$$D = \sum_{k=1}^{H-1} \kappa_i^k + d^H \qquad (4.4)$$

Equation 4.4 is the essence of preemptive cut-through. Preemptive cut-through allows the message to cut through a network switch since the message is sent out before it is fully received. It uses preemption to provide real-time message transfers over multiple links. It also is immune to the effects of a message arriving into a switch in several pieces. As long as the messages are held for a sufficiently long time, cut-through can be achieved even if message transmissions are preempted on the previous link.

We have described a new method of message transport for real-time messages, and a restriction on the scheduling of real-time messages using this method. We now prove that this new method satisfies our original goal of low-latency real-time communication.

Theorem 4.2 (Preemptive cut-through Minimum end-to-end latency) *Preemptive cut-through can provide the minimum possible end-to-end guaranteed latency τ for*

message stream M_i.

Proof: To achieve the minimum possible delay for message stream M_i, link delay bounds $\{d_i^1, d_i^2, \ldots, d_i^H\}$ must be set to their minimum. If $d_i^j = \tau_i$, $\kappa_i^j = 0 \forall\ \ell^j \in \Gamma_i$, then $D_i = \tau_i$. ■

This minimum latency achieved by preemptive cut-through is the same as in circuit-switched lines, and is much less than $H\tau_i$, the limit of previous approaches.

Up to this point, we have examined preemptive cut-through in an ideal context, ignoring processing delays and header sizes. In the real world, a small but finite amount of processing is needed before a message is accepted into a node. We represent this processing delay on a node as p. In addition, no data can be processed before the header is received. With a header transmission time τ_h, a minimum delay of τ_h must be added each time the message passes a new link. This factor can be added into the cut-through delay, resulting in an actual cut-through delay of $\kappa + \tau_h$. When these factors are incorporated into the idealized end–to–end delay equation, we obtain:

$$D = \sum_{k=1}^{H-1} (\kappa_i^k + \tau_h) + d^H + Hp \tag{4.5}$$

4.3 δ-stamping

The implementation of preemptive cut-through on an actual network requires the availability of r_i^{j-1} to compute the ready time r_i^j. Since globally synchronized clocks are impractical in large, geographically distributed networks, the ready time computation must not rely on global synchronization.

Piggyback timing has been used in several previous research efforts as a means to reduce jitter [53] or to change the playback time of data [17]. We propose a piggyback timing technique called *δ-stamping* to compute the ready time on a forward link. The technique works as follows (see figure 4.2): Denote the time on the clock on the node serving link ℓ^{j-1} as t^{j-1}. On ℓ^{j-1}, when the header of the message is sent out, the time difference between when the message could have first been sent, and the actual sending time $\delta t_i^{j-1} = t^{j-1} - r_i^{j-1}$ is "stamped" on the header. When the header is received at the node serving ℓ^j, the ready time there is computed as $r_i^j = t^j + \kappa_i^{j-1} - \delta t_i^{j-1}$. To preserve schedulability, the incoming portions of the message are held until r_i^j. This interval is called the *synchronization hold*. At r_i^j, the scheduler is informed of the arrival of m_i. The deadline z_i^j is computed and the message is scheduled for departure.

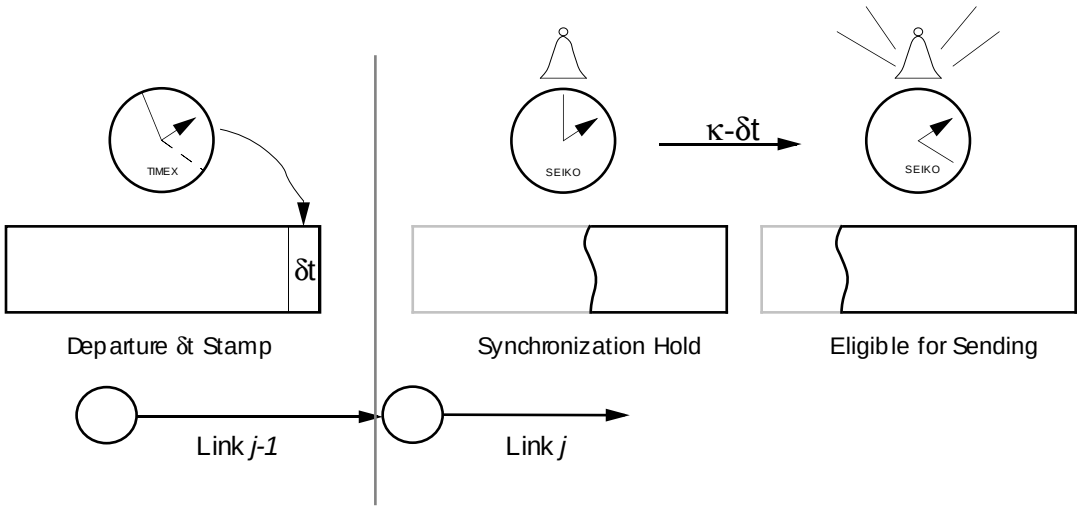


Figure 4.2: δ-Stamping

Examining the δ-stamp operation in detail, it can be seen that no absolute time

values are passed between two nodes on the network. If propagation delays are constant on links, the δ-stamp operation will in effect provide the offset necessary to synchronize the two clocks. To show that δ-stamping does indeed work, we calculate r_i^j assuming that the clocks on the nodes serving these links are different. In the following, the message index i has been dropped for clarity. On ℓ^{j-1}:

$$\delta t^{j-1} = t^{j-1} - r^{j-1}$$

On ℓ^j,

$$\begin{aligned} r^j &= t^j + \kappa^{j-1} - \delta t^{j-1} \\ r^j &= r^{j-1} + \kappa^{j-1} + t^j - t^{j-1} \end{aligned} \tag{4.6}$$

With a global clock, $t^j = t^{j-1}$, resulting in $r^j = r^{j-1} + \kappa^j$, which matches theorem 4.1. With non-global clocks, δ-stamping eliminates the difference in clocks.

Another major benefit of δ-stamping is the elimination of jitter on the message transfer latencies on intermediate nodes. A similar technique was first proposed by Verma [53]. We show that δ-stamping achieves the same effect.

Theorem 4.3 (Jitter reduction with preemptive cut-through) *With preemptive cut-through, ignoring processing delays, the end–to-end delay-jitter is guaranteed to be* $J_i \leq \kappa_i^H$.

Proof: By definition, $J_i = \max(D_i) - \min(D_i)$. The maximum value is given by (4.4) as $\sum_{k=1}^{H-1} \kappa_i^k + d_i^H$. To find $\min(D)$, we note that on the path of M_i, the

cumulative delay on links $\ell^1, \ell^2, \ldots, \ell^{H-1}$ is $\sum_{k=1}^{H-1} \kappa_i^k$ which is constant.

On ℓ^H, the minimum delay is τ_i since the message may be transmitted immediately upon arrival. Thus $\min(D) = \sum_{k=1}^{H-1} \kappa_i^k + \tau_i$. As a result, $J_i = d_i^H - \tau_i = \kappa_i^h$. ■

This result is in general agreement with Verma [53]. In fact, a smaller value for J cannot be obtained with an earliest deadlined first scheduler. However, since $\sum_{k=1}^{H-1} \kappa_i^k$ is less than the end-to-end latency achieved in [53], $\max(D)$ and $\min(D)$ are much less with preemptive cut-through. Thus, preemptive cut-through provides lower end–to–end delays and with the same jitter as previously offered real-time message transfer mechanisms.

4.4 Real-Time Message Transfers with ATM

Broadband-ISDN uses the Asynchronous Transfer Mode (ATM) [15] as its data transfer mechanism. ATM is a packet switching data transfer protocol, designed to rapidly switch small packets called *cells*. The ATM protocol is implemented on ATM switches. This discussion assumes an ATM switch model based on output buffering [60]. The functional block diagram of an ATM switch is shown in figure 4.3. In the switch, a cell traverses the following blocks:

Input Processing: The cell type is identified from its Virtual Channel/Virtual Path identifier. The arrival time is recorded. VCI Translation performed.

Switching: The cell is routed through a switch to its output link.

Cell Queueing/Admittance: Cell is inserted into an output queue in delay priority order.

Link Scheduling: The link scheduler controls link usage by allocating link time to real-time and delay-insensitive message streams.

On each output link, ATM maintains quality of service by separating data into different queues such as real-time (traffic sensitive to both delay and loss), delay sensitive and loss sensitive data queues. Within each queue, delay and loss are controlled by assigning priorities [61–63] to cells. For interactive real-time message streams, losses are unacceptable. Therefore a fixed buffer space is always reserved for real-time cells. This reduces the QOS maintenance of interactive real-time message streams to the control of maximum delay on each link.

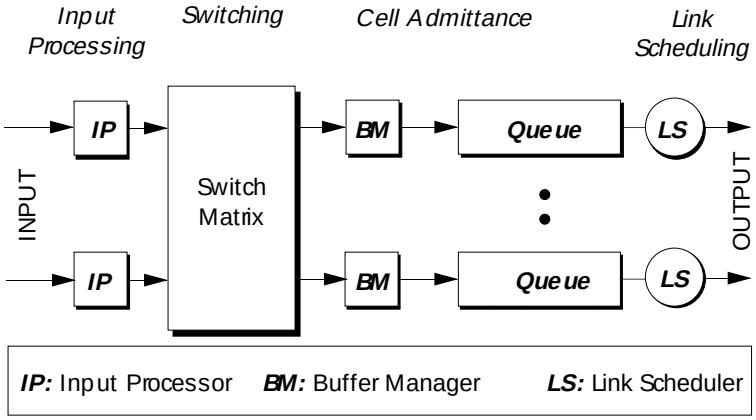


Figure 4.3: ATM QOS maintenance mechanisms

Real-time applications use *messages* to exchange information. A message is the smallest unit of self-contained data exchanged between applications. A portion of

digitized audio, the I-Frame of MPEG compressed video [64], or a full radar reading from a military installation can be considered as message examples since they are all self-contained. Conversely, an ATM cell cannot be considered a message as it is too small to carry data that can be self-contained.[1]

A message represents one action or event, so, it has one end–to–end delay requirement. When data is transferred by ATM, the message is broken down to cells and the relationship between the parts of the message is lost. Since ATM cannot identify messages,[2] it has to use *cell-level* quality of service maintenance. With cell-level QOS maintenance, each cell arriving to a node is assigned a deadline. More importantly, since the relationship between cells is not known, the quality of service for each cell is individually maintained.

The total delay experienced by a cell on an ATM switch consists of the queueing delay w and the processing delay p. As link capacities increase and the increased capacity is utilized, w will decrease. However, the the number of cells in the switch at one time will remain the same. For example, using an *M/D/1* queue approximation, with link capacity μ cells/sec and arrival rate of λ cells/sec, and $\rho = \lambda/\mu$, $w = \rho/(2(1-\rho))$ [13]. Suppose the link capacity and usage are both increased by a factor $\beta > 1$; $\mu_\beta = \beta\mu$ and $\lambda_\beta = \beta\lambda$. The waiting time becomes $w_\beta = (1/\beta)w$. In both cases, the number of elements in the switch waiting for transmission remains the same. This value, by Little's Law [13] $N = \lambda w$, is $N = \lambda\rho/(2(1-\rho))$.

---

[1] Cells carrying telephone conversations are one exception. The ATM standard was designed so that low-fidelity audio packets can fit in one cell.

[2] One may presume that the VCI/VPI identifiers can be used to identify message. However, they only represent the message stream and cannot be used to distinguish individual messages in the stream.

The processing overhead for delay control consists of fixed factors such as cell identification and variable factors proportional to the number of items with which the delay parameters are compared. As link capacities increase, unless p decreases by the same proportion, its effect will become more significant in the total end–to–end delay. If the variable factors decrease, the processing delay will decrease as well. The *end–to–end processing requirement* of a message is the total QOS maintenance processing by all ATM link controllers on the message path. As link capacities increase, the total processing requirement will also increase in proportion, requiring very high speed link processors.

4.4.1 Message Level QOS Maintenance in ATM

We propose a *message-level* delay control scheme as an alternative to cell-level control to overcome the challenges posed by increasing link capacities. The idea of using message-level Quality of Service was also proposed by Turner [65]. However, his approach only considers buffer allocation, not delay control. With our scheme, instead of assigning deadlines to cells, they are assigned to messages. Once the deadline is computed for the message, all other cells belonging to the message inherit the same deadline without any other computation, regardless of when they arrive. Since one deadline represents multiple cells, the scheduler only stores one deadline entry per message instead of one entry per cell. This reduces the search space for deadline comparisons.

Most critical or interactive real-time data streams carry digital representations of analog signals. Technological developments have increased the accuracy of this

representation in two orthogonal directions. One direction is the rate axis, where the sampling rate of data is increased to increase the frequency range of the sampled analog signal. For example, voice from telephone lines is digitized at an 8KHz sampling rate, to represent a 3.6KHz audio bandwidth. On the other hand, Compact Discs (and digital high-fidelity audio broadcasts) sample music at 44KHz, reproducing a 20KHz audio spectrum. The second direction of accuracy increase is the sample size, which increases the fidelity of the representation. Using the same example, phone systems use 8-bit samples with a 48-decibel dynamic range (ignoring companders) while Compact Discs use 16 bits to reproduce sounds at the full 96-decibel dynamic range of the human ear.

In current technological developments, there is a noticeable trend in an increase in sample size instead of sampling rates. This is most visible in video, where the 30 frames-per-second sampling rate is maintained while the resolution of the frame is increased considerably. Although this increase in the sample size increases the total amount of data generated, it does not increase the message generation rate. The same number of messages are generated, but the message size increases. This is significant if only cell-level QOS maintenance is used on broadband networks. With increased sample resolutions, the number of cells introduced to the network will increase as well, resulting in even longer processing delays at switches. If only messages are considered, as long as the number of messages remain the same, an increase in message size will not affect the processing delays at switches.

In chapter 4, the preemptive cut-through message transfer method was shown to provide low end–to–end delays. Preemptive cut-through can provide message-level

QOS maintenance in ATM and still offer cell-level end–to–end delays. ATM does not allow the preemption or cut-through of cells;[3] there are no mechanisms to describe the preemption of a cell or its re-transmission. Still, ATM cells can be used as the smallest indivisible unit of the preemptive cut-through mechanism. In this case, the message is preempted only in an abstract sense; in reality the flow of data is simply stopped between cells.

To implement preemptive cut-through on an ATM switch, a message detection mechanism, an Earliest Deadline First scheduler [2, 57], and a message-departure time-stamping mechanism are needed. Our proposed preemptive cut-through link manager incorporating these functions is shown in figure 4.4. The link manager incorporates a timestamp-extraction ready-time assignment and cell routing unit at the input processing side. The data is stored in separate buffers, with a sharedbuffer for real-time message streams. The scheduler provides priority access to the link. For real-time messages, a δ-stamping unit at the output provides timestamping. This unit uses a reference clock on the node to obtain the time.

Message identification, δ-stamping and Storage

The message detection mechanism consists of special bits in the ATM cell header as well as a message detector on each input processor of the ATM switch. To detect message boundaries, the head and tail cells of the message need to be recognized. Other than the payload type bits, the ATM cell header does not provide any infor-

[3]Cell-level Cut-through has been proposed [66] but has not been accepted as a standard.

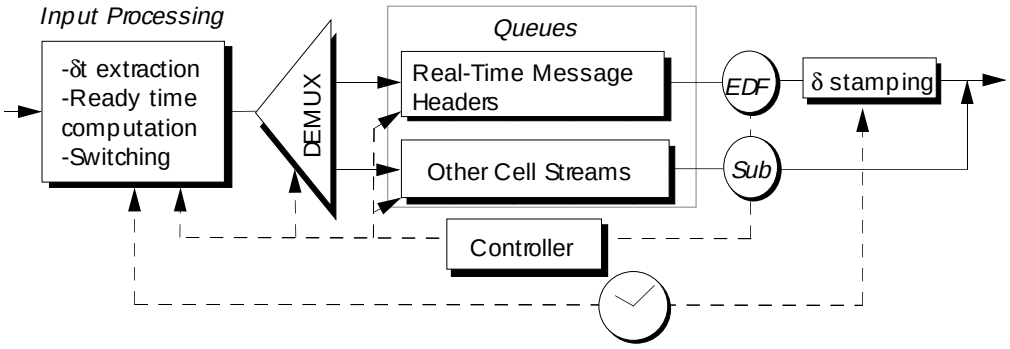


Figure 4.4: A preemptive cut-through link manager for an ATM switch

mation about the contents of the cell. We rename one of the two payload bits[4] the $M/\overline{C}$ (*Message/Cell*) to indicate that the cell is a boundary marker for a message. To distinguish between the header and the tail cell, the reserved bit in the ATM header is renamed as the $F/\overline{L}$ *(First/Last)* flag. The message detector uses these flags and a state variable to distinguish messages. The state diagram for the message detector, shown in figure 4.5 and listed below, consists of six states.

Seek: Initial state. If cell header contains $M/\overline{C} = 1$, and $F/\overline{L} = 1$, a transition is made to the *header* state.

Header: Message-related information such as the δ-stamp is extracted from the header and the message cell counter is reset. Any other cell arriving with the same VCI/VPI results in a transition to the *body* state. If two headers arrive without any other cells in the body of the message, a transition is made to the *error* state.

[4]The other bit indicates that the cell contains network related information.

Body: All incoming cells belonging to this VCI/VPI are counted. If the message tail cell with header flags $M/\overline{C} = 1$, and $F/\overline{L} = 0$ is received before the count reaches the message length, or if the count exceeds the message length, a transition is made to the *error* state. If the tail is received properly, a transition is made to the *tail* state. If a header is received, a transition is made to the *error* state.

Error: This state is for exceptions resulting from errors due to transmission failures. Since the message tail was not properly detected, several courses of action may be taken [65]: All the cells forming the message may be dropped. If only a few cells are missing, the message may be padded to its original length, or if the count was exceeded before the count was reached, a tail may be added to fix the message length.

Tail: This state is used to perform end of message operations such as counter reset. If a proper tail cell is detected, these operations are performed and a transition is made to the *seek* state.

Although preemptive cut-through will not drop cells due to congestion, cells may be lost due to transmission errors. Since message boundaries are used for scheduling, losing cells belonging to a message can result in a situation where the cells of another message are counted for an incoming message. Using a counter and first/last cell identifiers allows error checking for cell losses without significant overhead.

To reduce message processing times, messages must be accessed efficiently. Assuming a shared-buffer switch architecture [60], a two-level memory hierarchy stores

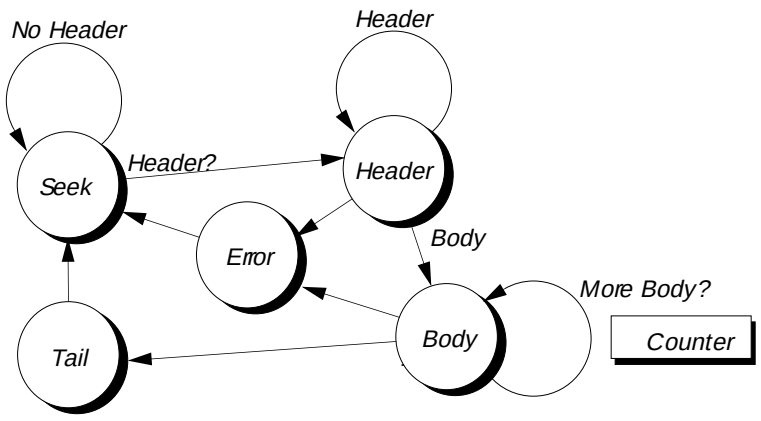


Figure 4.5: Message detector state diagram

messages. At the first level, a priority queue holds only message records, allowing rapid queue arrangement. The cells forming the message reside in the shared cell buffer (Figure 4.6).

4.4.2 The Preemptive Cut-Through Scheduler

The preemptive cut-through link manager hardware implements the earliest deadline first scheduler using a link controller and a priority queue. The link controller takes cells from the priority queue and transmits them. The priority queue keeps message records in deadline order. These message records point to the actual storage area of the cells forming the message (See figure 4.6).

For real-time messages, the priority queue insertion and dequeueing time determine the minimum processing delay p on an ATM switch. A priority queue can be implemented as a linked-list or heap [67] data structure. For very fast access to the queue, a content-addressable memory approach has been offered [68], however this

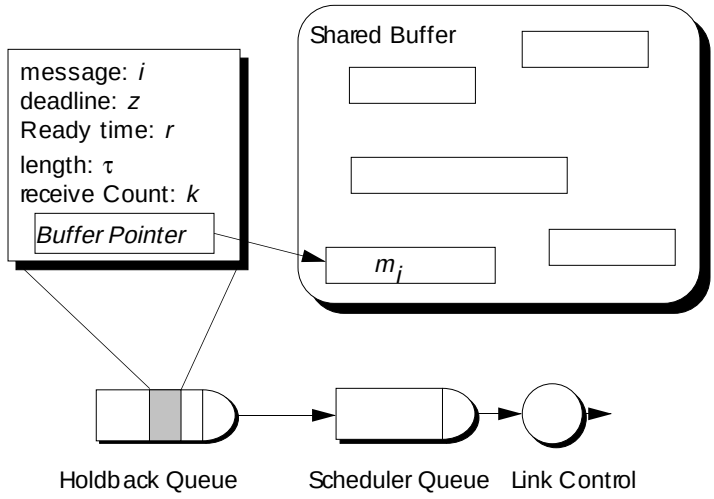


Figure 4.6: Message storage hierarchy for preemptive cut-through

is very costly. A link can transmit data at rates of gigabits/sec; therefore, it is imperative that the dequeueing time be bounded. Otherwise, the link capacity will be wasted by inserting idle cells when a cell is not available for transmission. The insertion time should also be low. In addition, since the burst insertion rate of a queue can exceed the output rate, multiple insertions should be able to progress simultaneously. A heap data structure is not amenable to parallel insertions. Other structures such as the SAPQ (Simultaneous Access Priority Queue) have been proposed [69], however, their processing delay characteristics are unknown. Therefore, the linked-list model is still the most appropriate for the priority queue.

A portion of the network traffic may not have strict delay or loss requirements. The preemptive cut-through link manager should provide link management to such streams as well. For most data traffic without real-time requirements, a bandwidth reservation (as opposed to a delay based check) is sufficient to maintain a prescribed

quality of service. This bandwidth can be computed by algorithmic techniques [70] or by using a pre-computed schedulable region [55]. The preemptive cut-through link manager reserves bandwidth for these non-real-time traffic classes using *sub-schedules*. The administration of the bandwidth is left to other schedulers optimized for the particular traffic class. These schedulers arrange the order of cells in sub-schedule queues according to the QOS maintenance policy in effect for the traffic class.

A *sub-schedule* is a mechanism to reserve bandwidth for traffic classes. The sub-schedules are considered together with the real-time messages during link reservation to guarantee the bandwidth reservation. The sub-schedule bandwidth B is reserved through periodic link usage requests with period T_{ss} and link usage time τ_{ss}, where $B = \tau_{ss}/T_{ss}$ indicates the percentage of the link bandwidth used for the sub-schedule. Both T_{ss} and τ_{ss} are in multiples of the cell transmission time τ_{cell}. As long as their ratio evaluates to B, T_{ss} and τ_{ss} can be set arbitrarily. However, the QOS requirements of the sub-schedule may limit T_{ss} since a longer period will increase the delay variation of cells.

Sub-schedules can be implemented by inserting a message record containing $c = \tau_{ss}/\tau_{cell}$ cells into the scheduler queue at periodic intervals of T_{ss}. Since only a record is inserted, sub-schedules do not incur significant overhead. The message record is assigned a deadline equal to the end of the period. This record is treated as any other real-time message record. When the record representing a sub-schedule is selected for transmission, the link controller sends cells from the sub-schedule queue instead of the real-time message buffer.

The sub-schedule concept is similar to the ATS [56] scheduler. However, ATS provides a fixed cycle time and varies the proportion of time allocated to four different classes. With sub-schedules, the total time is fixed as in ATS, but the sub-schedule may be not receive this time in one chunk. On the other hand, since the sub-schedule period can be arbitrarily set, different QOS levels can be maintained by using multiple sub-schedulers with different periods.

4.4.3 Processing Cut-Through Timing

Preemptive cut-through uses timing information contained in the incoming message header cell to determine the ready time of the message. The message detector decodes the δ-stamp in the cell and computes the time at which the message will be eligible for processing. The δ-stamp represents a time difference. Therefore, its maximum value is limited by the maximum delay bound on a link. If the bit clock of the link is used as a reference, a 32 bit δ-stamp field is sufficient to denote up to a 4-second delay with 10^{-9} sec precision. This should be sufficient for all practical purposes.

When a message is assigned the ready time r_i^j at time h_i^j, it must be held in a separate queue for the duration of the synchronization hold. Following an approach similar to the one offered by Kandlur [21] (see Figure 4.6) we use a *holdback queue* for this purpose. This queue is a priority queue, similar to the regular scheduling queue. However, the message headers contained in it are not seen by the scheduler. When a message in the holdback queue becomes ready for scheduling (when $r_i^j > t$), it is transferred to the scheduling queue.

During transmission, as the first cell of the message is sent, the current time on

the link is read and the cut-through time is computed. This value is stamped on the header cell. This provides the timing information for the next link.

4.4.4 Performance Analysis of Message-Level Delay Control

This section compares the performance of message-level and cell-level control mechanisms in terms of the total amount of processing and the incurred processing delay. Since the processing and delay are implementation dependent, abstract quantities for each basic operation are defined. Suppose on average, messages contain n cells and the priority queue contains q messages. The quantities listed in table 4.1 describe the delays or number of operations required for processing an arriving cell.

In table 4.2, the total number of operations per cell on each node and path for both a cell and for message-level quality of service maintenance are computed. The *processor speed multiplier* ξ converts operation counts to time. For example, if the queue insertion takes Q operations, it takes ξQ seconds.

As shown in the table, the total number of operations to send a message of n cells from its source to destination is $H(nPQ + \frac{1}{2}n^2qI)$ for cell-level control. For message-level control the total number of operations is $H(nPQ + \frac{1}{2}qI)$. While it is difficult to make a precise statement about these values, in all cases, the number of operations for cell-level control is significantly higher. The $PQ = TL+TS+BA+BC$ term represents essential operations, therefore it cannot be reduced any further. The queue insertion operation, which forms the bulk of QOS maintenance, can be reduced significantly with message-level control.

Table 4.1: Delays and operations performed on an arriving cell

Activity	Symbol	Description
		Delay Parameters
Header Arrival	HA	Duration of cell header arrival.
Delta Stamp	DS	Duration of delta stamp field arrival.
Buffer Copy	BC	Duration of cell copy to buffer location. $BC = \tau_{cell}$.
		Processing Operations
Table Lookup	TL	Number of operations to lookup the VCI/VPI. TL is constant.
Time Stamping	TS	Number of operations to compute arrival time. TS is constant.
Buffer Allocation	BA	Number of operations to allocate buffer space. BA is constant.
Queue Insert	Q	Number of operations to insert cell or message record into priority queue. Denoting the each link traversal with, I, on average, for message-level control, $Q_{msg} = \frac{1}{2}qI$, and for cell-level control $Q_{cell} = \frac{1}{2}nqI$ operations.

Message-level delay control reduces the effects of processing delay on the end–to–end delay. The cells of a message are pipelined through the links. The resulting end–to–end delay expression for cell-level control is:

$$D_{cell} = \sum_{k=1}^{H}(w_{cell}^{k} + \tau_{cell} + p_{cell}) + (n-1)\tau_{cell} \tag{4.7}$$

where p_{cell}, w_{cell}^{k} and τ_{cell} denote the processing time for the cell, the maximum waiting time[5] on the switch before receiving service, and the cell transmission time on the

[5]Assume that the ATM scheduler for link j is capable of guaranteeing the transmission of the cell within $w_{cell}^{j} + \tau_{cell}$ time units. The transmission can start any time within this interval, but must be completed before the interval ends. Without this guarantee, ATM cannot provide real-time

Table 4.2: Performance comparison of cell and message-level delay control

Action	Cell-Level Operations	Message-Level Operations
First Cell		
Table Lookup	TL	TL
Time Stamp	TS	TS
Buffer Allocate	BA	BA
Buffer Copy	BC	BC
Total Prequeueing Operations	PQ	PQ
Queue Insert	$\frac{1}{2}nqI$	$\frac{1}{2}qI$
Total Processing affecting End–to–End Delay on a link	$PQ + \frac{1}{2}nqI$	$PQ + \frac{1}{2}qI$
Total Processing affecting End–to–End Delay on a path	$H(PQ + \frac{1}{2}nqI)$	$H(PQ + \frac{1}{2}qI)$
Other $(n-1)$ cells		
Prequeueing operations	nPQ	nPQ
Queue Insert	$\frac{1}{2}(n-1)nqI$	-
Total Processing for a message on a path	$H(nPQ + \frac{1}{2}n^2qI)$	$H(nPQ + \frac{1}{2}qI)$

link, respectively. In (4.7), the summation and the processing delay terms describe the delay experienced by the first cell. It is assumed that the remaining cells are sent after the first cell without any delay.

The end–to–end delay expression for message-level control was given in equation 4.5. Substituting $\tau = n\tau_{cell}$, $\kappa = w_{msg}$, $d = w_{msg} + n\tau_{cell}$, $\tau_h = \tau_{cell}$ and using msg indices for message-level time parameters, we obtain:

$$D_{msg} = \sum_{k=1}^{H}(w_{msg}^{k} + \tau_{cell} + p_{msg}) + (n-1)\tau_{cell} \tag{4.8}$$

If the same real-time scheduler is used, $w_{msg} = w_{cell}$.

message transfers.

The ratio of processing delays can be computed using results from table 4.2. As shown in Figure 4.7, some operations are overlapped with the arrival of the cell and copying to the buffer. The delay expression for cell-level control is

$$p_{cell} = \max(HA + \xi(TL + TS + Q), HA + \max(\xi(TL + BA), BC) + BC) \quad (4.9)$$

For message-level control,

$$p_{msg} = \max(HA + \max(DS, \xi TL) + \xi TS + \xi Q, HA + \max(\xi(TL + BA), BC) + BC) \quad (4.10)$$

For a 1 gigabit/sec link, $BC = 424\,\text{ns}$, $HA = 40\,\text{ns}$ and $DS = 32\,\text{ns}$. Assuming that $\xi TL = \xi TS = \xi BA = 50\,\text{ns}$ and $\xi I = 20\,\text{ns}$, p_{cell} and p_{msg} are tabulated for various values of n and q in table 4.3. From the table, it can be seen that, with the values chosen, p_{msg} is usually less than the time taken to copy cells within the switch. Comparing cell and message-level delays, for most cases $p_{cell}/p_{msg} \gg 1$.

The values used in the above analysis are only intended to show that message-level processing delay is usually less than delays incurred using cell-level processing. The numbers we have used to obtain these results are projected estimates of hardware delays. As far as we know, these parameters have not yet been published for any particular high-speed switch. When they are available, this comparison can be validated with actual values.

Table 4.3: p_{cell} vs. p_{msg} for various queue lengths

q	n	p_{cell} (ns)	p_{msg} (ns)	p_{cell}/p_{msg}
1	10	848	848	1.00
	100	1145	848	1.35
	1000	10145	848	11.96
10	10	1145	848	1.35
	100	10145	848	11.96
	1000	100145	848	118.09
100	10	10145	1170	8.67
	100	100145	1170	85.59
	1000	1000145	1170	854.82

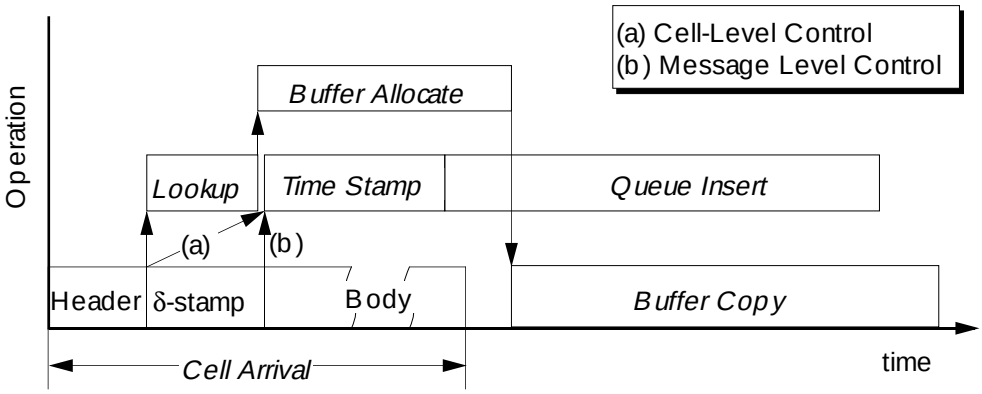


Figure 4.7: Processing delay overlap

4.5 Discussion

This chapter presented a new message transfer mechanism termed preemptive cut-through for real-time message transfers on point–to–point or wide-area networks. In the limit, preemptive cut-through was shown to provide the lowest possible end–to–end delay that can be achieved with any means, including circuit-switching. While overheads will undoubtedly make this unachievable in practice, the technique can still

offer very low delays compared to other packet-switched methods.

The preemptive cut-through message transfer mechanism employs Earliest Deadline First Scheduling [57]. Compared to previous approaches using the same scheduler [2, 21, 53], preemptive cut-through provides lower end–to–end delay bounds and lower jitter. If the network does not allow cut-through, the delay bounds are similar. However, the processing time is still significantly less since message-level QOS management is not available in previous approaches. Preemptive cut-through also provides significantly lower end–to-end delays (τ vs. HT) and lower jitter compared to non-scheduler based methods such as Stop-and-Go Queueing [51].

We presented a message-level delay control scheme for ATM to transfer interactive real-time messages over a wide area or point-to-point network. The scheme provides hard delay bounds while reducing overheads associated with cell-level delay control. The overhead is reduced by adding information onto the cell header. Since the standard header structure is preserved, this scheme can co-exist with cell-level control schemes for other traffic classes.

Preemptive cut-through is used for message-level delay control. Although preemptive cut-through requires more buffer space than a cell-level control scheme, this requirement is not significant when memory is plentiful and inexpensive. The benefits of additional buffering are the decreased jitter-reduction operations at the destination node and reduced the computational requirements for link processing. Thus, less expensive processors can be used for link control and jitter reduction, or one processor can control several links.

The sub-schedule mechanism manages bandwidth for message streams without

stringent QOS requirements. Although other time sharing schedulers such as ATS [56] also provide bandwidth management, they fail to guarantee delays for interactive streams.

Some of the ideas presented here are useful in other parts of ATM Quality of Service Maintenance. Detecting messages in the cell stream is also useful for congestion control of non-real-time message streams. When buffer space is needed, Turner [65] proposes to discard all cells belonging to the same message, but does not provide a means to identify messages in a cell stream. Our message detection mechanism can be used to implement Turner's approach.

Chapter 5

Adaptive Message Stream Admission

Before a real-time message stream is allowed into a network, the network server must verify that the stream's QOS requirements can be met. The procedure of verifying QOS requirements and allocating network resources is called *message stream admission*.[1] Figure 5.1 presents a proposed method of message stream admission [2].

There are several criteria by which a message stream admission procedure can be judged. For users of the network, the best procedure is the one that admits the most streams. If there are streams that are more "difficult" to serve (such as streams which require low end-to-end latencies, or which would result in high link utilizations), a procedure admitting more of these streams is preferred. For the network server, maximum utilization of network resources is the most important consideration. For both, the cost (in terms of both computation time and hardware) for stream admission is not terribly important. This is because each message stream consists of a large

[1] The same procedure has also been referred to as "Channel Establishment" [2, 21] and "Admission Control" [17].

Table 5.1: Stream admission procedure, adapted from [2]

1. **Routing:** Determine the set of links that messages will follow from the source to the destination of M_i. While this is a very important step, we assume in this paper that a path Γ_i, for the message stream is already known; routing will not be discussed further.
2. **Link Delay Bound Computation:** For each $\ell^j \in \Gamma_i$, compute the minimum d_i^j using an algorithm to find the minimum delay bound $d_i'^j$ [22].
3. **End-To-End Latency Verification:** Compute the minimum D_i'. If $D_i' \leq D_i$, then M_i may be admitted to the network.
4. **Post Admission Adjustment:** Distribute excess slack for the message stream. Ferrari [2,21], suggests that if the message stream is admitted, the server should divide the end–to–end slack $S_i = D_i - D_i'$ along the path, resulting in link completion intervals $d_i^j \leftarrow d_i^j + \frac{S}{H}\ \forall \ell^j \in \Gamma_i$. Zheng [22] offers a similar solution.
5. **Notification:** Transmit all d_i^j to the nodes that serve the links in Γ_i, in preparation for scheduling the first message of M_i.

number of messages for most real-time applications; the one-time cost is amortized over the total number of messages in the stream.

In communication networks, link traffic loads obviously are not static. This is due to the unpredictable pattern of network usage. TYMNET [13] is an example of a network whose admission procedure adapts to changing network loads to consistently maintain high utilization. This network is based on virtual circuits for statistical message streams, so the procedure cannot be directly used for our purpose. However, we believe that in the long run an adaptive procedure will be superior to a static real-time stream admission procedure. The following example illustrates why on a small scale:

Example 1: Suppose message streams M_1 and M_2 of figure 3.1 with parameters (τ, T, D, Γ) = (5,20,12, $\{AC,CD,DE\}$) for M_1 and (6,18,15, $\{BC,CD,DF\}$) for M_2 were admitted using the stream admission procedure of Figure 5.1 and the Preemptive Cut-Through delay expression of equation 4.4. To admit M_3 with parameters (3,9,14, $\{AC,CD,DG\}$), the same admission procedure is applied. At this point, M_3's request for admission is rejected. The results are as follows:

parameter	M_1	M_2	M_3
d'	$(5,5,5)$	$(6,11,6)$	$(8,14,3)$
D'	5	11	19
S	7	4	-5
d	$(7,7,8)$	$(7,12,8)$	–

M_3 is rejected because $D' > D$.

In the example, the rejection of M_3 is puzzling because the overall network utilization is not very high. If the network server instead assigned all of the slack for message streams M_1 and M_2 to link CD, d_1^{CD} and d_2^{CD} would be increased. In that case, it becomes possible to reduce d_3^{CD} and M_3 could be admitted to the network.

A request for admission of stream M_i can fail for one of two reasons: (i) $D' > D$ as shown in example 1; or (ii) The addition of M_i to some $\ell^j \in \Gamma_i$, renders ℓ^j unschedulable. In the first case, failure can be prevented by reducing $d_i'^j$'s on one or more of $\ell^j \in \Gamma_i$. This reduction is impossible if the admissions procedure is non-adaptive. In the next section, we present an adaptive stream admission procedure which admits a larger number of real-time message streams. This procedure relies

upon an algorithm for delay bound reduction (DBR).

5.1 Adaptive Stream Admission

Real-time message streams require hard bounds on the end–to–end delivery delay. However, they are insensitive as to how this delay is divided within the network. The adaptive stream admission procedure takes advantage of this insensitivity to admit a larger number of real-time message streams. The procedure works by re-adjusting the link delays of active message streams. To do so, it needs to keep track of which delays can be adjusted. Initially, minimum d_i^j are assigned to all $\ell^j \in \Gamma_i$. This results in messages arriving early to the destination node. In the destination, the messages are delayed until their delivery time. This delay is called the *end–to–end slack* S_i of M_i.

Adaptive stream admission tries to distribute S_i to admit failed admission requests. As needed, the S_i are decreased and assigned to one or more links on the path of the new message. This increases the delay bounds of the existing streams. When the delay bound of a stream is increased, a new message stream can be added on a link more easily.

For adaptive stream admission to function, it needs a procedure to decrease the link delay bounds of a new message stream after slack is added to existing messages. It also requires a means to distribute slack. We define several preliminaries that will be used in developing this procedure. In the following, time is assumed to be in discrete. The unit of time is the transmission time of a single bit.

Definition 5.1 (Full period count) *Given M_i, T_i, a time interval $[0,t]$ and the starting time of a period s_i. If $s_i + T_i \leq t$ then there exists at least one* full period *of M_i in $[0,t]$. Assuming that the first period starts at $s_i = 0$, the* full period count *$F_i(t)$ is the total number of full periods of M_i in $[0,t]$. The full period count excludes the last period in $[0,t]$. Formally,*

$$F_i(t) = \left\lfloor \frac{t-1}{T_i} \right\rfloor$$

Definition 5.2 (Residual period count) *Given M_i, T_i, a time interval $[0,t]$ and the starting time of a period s_i. If $s_i + T_i > t$ then this is a* residual period *of M_i. Only the last period in $[0,t]$ which starts at $s_i = F_i(t)T_i$, can be a residual period. If $[0,t]$ includes the active portion of the period, which is between $[F_i(t)T_i, F_i(t)T_i + d_i]$, then the* residual period count *$R_i(t)$ is set to count the message transmission. Formally,*

$$R_i(t) = \begin{cases} 1 & \text{if } F_i(t)T_i + d_i \leq t \\ 0 & \text{otherwise} \end{cases}$$

The full period count and residual period count can be used to compute the total active time of a link:

Definition 5.3 (Link Usage Time) *The* link usage time *on ℓ^j, $L^j(t)$, is total time ℓ^j is needed for transmission by all $M_i \in \Omega^j$ in a time interval $[0,t]$. It can be computed as:*

$$L^j(t) = \sum_{M_i \in \Omega^j} \left[F_i(t) + R_i(t)\right] \tau_i \tag{5.1}$$

With the above definitions, the schedulability of a link can be formally defined:

Lemma 5.1 (Schedulability of a Link) *The set of messages Ω^j are schedulable on ℓ^j by an earliest deadline first scheduler if and only if the (i) link utilization $\sum_{M_i \in \Omega^j}(\tau_i/T_i) \leq 1$ and (ii) $L^j(t) \leq t \; \forall t$.*

Proof: (i) The utilization condition states that a link cannot be used beyond its capacity. (ii) For the link to be schedulable, the total time data is transmitted on the link must be less than the t; otherwise, this would indicate that there is more data to send than time to send it. Since messages repeat the same pattern after the period $[0, \text{lcm}(T_i \in \Omega^j)]$, it is sufficient to check for $L^j(t) \leq t$ in $\forall t \in [0, \text{lcm}(T_i \in \Omega^j)]$. ■

It has been shown that the value of $L^j(t)$ only changes at the message deadlines [22, 71]. Therefore, it suffices to check points $S = \{t : t = nT_i + d_i; i \in \Omega^j; n = 0, 1, \ldots, \text{lcm}(T_i \in \Omega^j)/T_i\}$ to assure schedulability.

Definition 5.4 (Minimum Delay Bound) *The minimum value of d_i^j such that $L^j(t) = t$ for some $t \in S$ is called the* minimum delay bound*, $d_i'^j$.*

Corollary 5.1 *If $L^j(t) < t$ for all $t \in S$, then d_i^j can be reduced by $\min(t - L(t)) \forall \, t \in S$.*

Definition 5.5 (Deadline at t) *Given a time instant t, the deadline z corresponding to the period in which t falls is given by $z_i(t) = F_i(t)T_i + d_i$.*

The *delay bound reduction (DBR) algorithm* reduces the minimum delay bound d'^j_n of a selected message stream $M_n \in \Omega^j$. All delay bounds $d^j_i = d'^j_i \ \forall M_i \in \Omega^j$ i.e. have been minimized, using [22]. In order to further reduce d'^j_n, the order of message transmissions must be altered. This order is determined by the deadline z^j_i of each message. Therefore, the deadline of m_n must be moved before one or more deadlines z_i for each possible message arrival pattern. Intuitively, the delay bound reduction algorithm is a search to find all deadlines falling in the period of m_n that can be increased such that the message transmission order is changed. Let the maximum amount of increase for message stream i on link j be symbolized by α^j_i. α^j_i cannot exceed $T_i - d^j_i$ and is also bound by the total slack S_i; in other words, $\alpha^j_i = \min(S, T_i - d^j_i)$. All valued computed using these increased deadlines are denoted by the tilde ($\sim$).

We prove the necessary conditions for DBR in theorem 5.1.

Lemma 5.2 *Given ℓ^j with message streams Ω^j. The delay bound d^j_n of M_n can be reduced if there exists α^j_i such that $z_i(t) < z_n(t) \leq z^j_i(t) + \alpha^j_i \ \forall t \in S_k = \{t : nT_k + d'^j_k; n = 0, 1, \ldots, \mathrm{lcm}(T_i \in \Omega^j)/T_k\}$*

Proof: For each $t \in S_k$, if $z_i(t) < z_n(t) \leq z^j_i(t) + \alpha^j_i$, then m_n will be transmitted before m_i. If there are no such $z^j_i(t)$, then since the order of transmission cannot be changed, $z^j_n(t)$ cannot be reduced. Thus d^j_n cannot be reduced. ■

Lemma 5.3 *If there exists α^j_i such that $z^j_i(t) < z^j_n(t) \leq z^j_i(t) + \alpha^j_i = \tilde{z}^j_i(t)$ for the given t, $z_n(t)$ can be moved back in time by τ_i.*

Proof: Since m_n is sent before m_i, m_n need not wait for the transmission of m_i. The duration of this transmission is τ_i. Thus $z_n(t)$ can be moved back in time by τ_i. ■

Theorem 5.1 *The delay bound $d_n'^j$ of M_n can be reduced if there exists α_i^j such that (i) $z_i^j(t) < z_n^j(t) \leq z_i^j(t) + \alpha_i^j = \tilde{z}_i^j(t)$ $\forall t \in S_k = \{t : rT_n + d_n'^j; r = 0, 1, \ldots, \text{lcm}(T_i \in \Omega^j)/T_n\}$ and (ii) $\alpha_i^j \geq \tau_n$.*

Proof: The first condition was proven in lemma 5.2. We prove the second condition by contradiction. Assume that d_n^j is reduced by τ_i and $d_i'^j$ is increased by $\alpha_i^j < \tau_n$. Then for any t, $\tilde{L}(\tilde{z}_i(t)) \leq \tilde{z}_i(t)$ must hold. Since $d_i^j = d_i'^j$, there exists at least one $z_i(t)$ such that $L(z_i(t)) = z_i(t)$. After $z_n(t) < \tilde{z}_i(t)$ is chosen, $\tilde{L}(\tilde{z}_i^j(t)) = L(z_i(t)) + \tau_k$ since m_n will be transmitted before m_i. To perserve schedulability at $\tilde{z}_i^j(t)$, $\tilde{L}(\tilde{z}_i^j(t)) \leq \tilde{z}_i^j(t)$ must hold. However, since $\alpha_i^j < \tau_k$, $\tilde{L}(\tilde{z}_i^j(t)) < L(z_i(t)) + \tau_k$, resulting in $\tilde{L}(\tilde{z}_i^j(t)) > \tilde{z}_i^j(t)$ which contradicts the schedulability assumption. ■

Theorem 5.1 showed the conditions required to reduce a delay bound. The amount of reduction in $d_n'^j$ depends on how many z_i are expanded such that $z_i^j(t) < z_n^j(t) \leq z_i^j(t) + \alpha_i^j$. To compute the reduced delay bound, the following steps are carried out:

1. **Expansion:** Temporarily expand d_i^j $\forall M_i \in \Gamma_j$ except for M_{sel}, such that $\tilde{d}_i^j = d_i^j + \alpha_i^j$.

2. **Minimization:** Minimize $d_n'^j$ using an optimal minimum delay bound determination algorithm such as [22].

3. **Contraction:** Minimize $\tilde{d}_i^j$ to recover unused slack, by applying the minimum delay bound algorithm to each expanded stream.

We can illustrate the use of DBR on example 1. Ignoring step 4 of the non-adaptive admission procedure, save the slacks of M_1 and M_2. Thus $d_1 = d'_1 = (5,5,5)$ and $d_2 = d'_2 = (6,11,6)$ and the slacks $S_1 = 7$, $S_2 = 4$ are saved at the destination nodes. This allows the reduction of d_3^{CD}. With this reduction $d'_1 = (5,5,5)$, $d'_2 = (6,14,6)$ and $d'_3 = (8,8,3)$. S_2 is reduced to 1, showing that slack was consumed from S_2 to reduce d_3^{CD}. Now, all delay bounds of M_3 are less than the period and M_3 meets the for end-to-end delay bound.

The DBR algorithm can be incorporated into the message stream admission procedure of Table 5.1; the result is the procedure of figure 5.1. This admission procedure is adaptable to changing network conditions. DBR is used to: (i) attempt to make reduce d_i to be less than T_i; and, (ii) reduce D'_i for a stream M_i that is schedulable on each link but with $D'_i > D_i$.

To make a message schedulable on links, the DBR algorithm must be applied to each $\ell^j \in \Gamma_{new}$ where the new stream M_{new} is unschedulable. To reduce D'_{new}, DBR must be applied to some $\ell^j \in \Gamma_{new}$ (see Figure 5.1) as well. In both cases, the order with which the links are processed is important. If a relatively uncongested link is chosen first, d_i^j may be reduced so much as to forego the need to examine other links. On the other hand, there may be several M_i such that $\Gamma_i \cap \Gamma_{new} \neq \emptyset$. In this case, if one of the shared links is examined first, the slack of the streams are expended. When the other shared links are processed, there will not be much slack left to decrease d'_{new}. Since there are so many considerations, we chose to randomly select the links to be

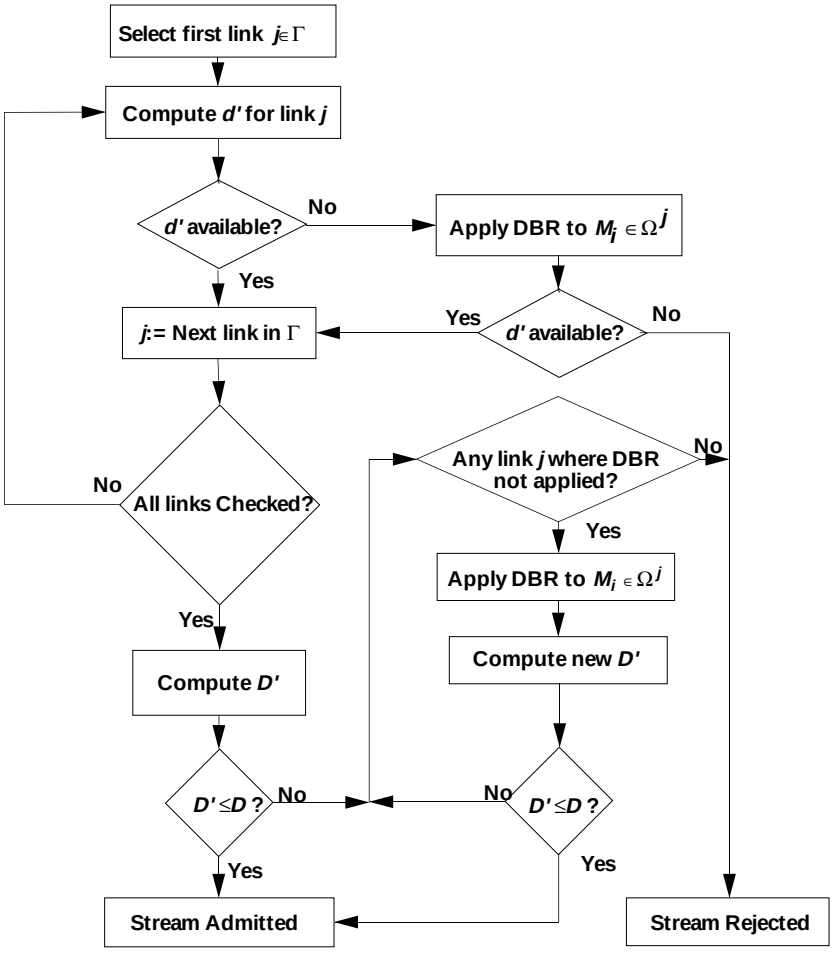


Figure 5.1: Adaptive stream admission procedure

processed. If our method can be shown to be effective with a random link processing order, other optimizations can be used to increase its effectiveness in the future.

5.2 Performance Analysis of Adaptive Stream Admission

We evaluated the effectiveness of the DBR algorithm with an experiment. In this experiment, the stream admission procedure shown in Table 5.1 was compared to the Adaptive Stream Admission Procedure of Figure 5.1. Both procedures were used to admit a set of real-time message streams to a network. The network modeled for this experiment was an early version of the ARPANET computer network; see Figure 5.2. This network has 56 nodes and 135 links. The average number of links per node is 2.4, and the maximum is 4.

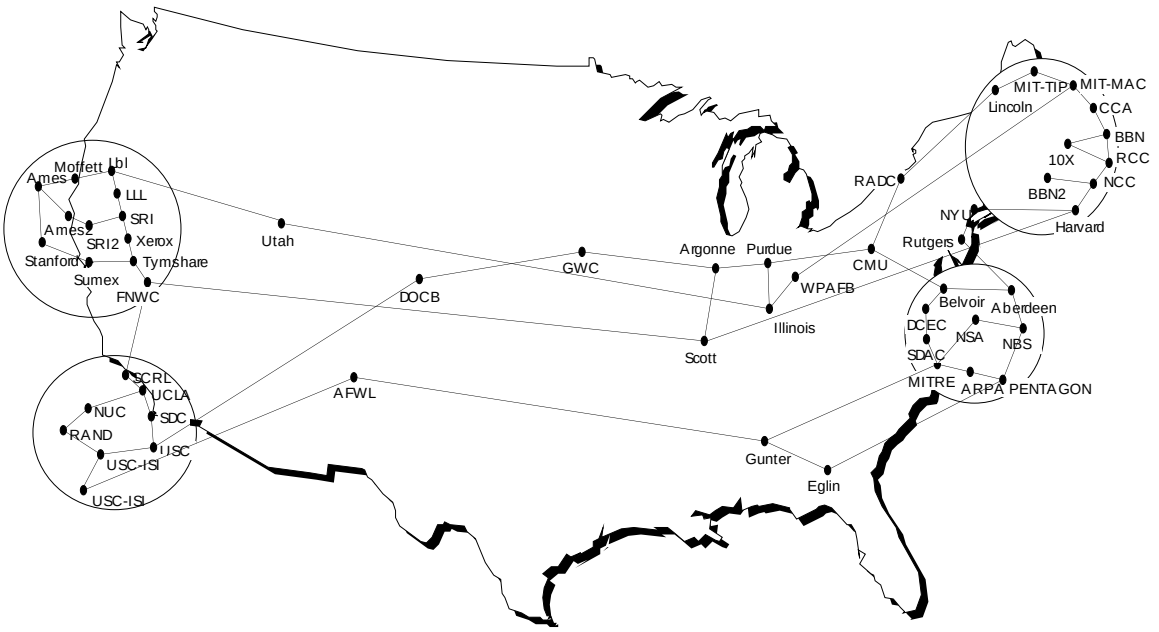


Figure 5.2: Test network for adaptive stream admission (Adapted from [1])

The same set of message streams was applied to the network in both cases. Each

set consisted of 1000 message streams. The source and destination of each message stream were randomly selected from the nodes in the network. Streams with adjacent source and destinations were not considered. Each stream was routed through the shortest distance path using the Floyd-Warshall algorithm [72]. The message transmission time and period were set to $\tau_i = 500$ and $T_i = 10000$. This set the utilization for each message stream to 5% on each link. The end-to-end latency requirements of streams were varied as follows: For each message stream, $D = \psi HT$, where ψ is a uniformly distributed variable within a specified range. A small value for ψ represents very tight latency requirements on the message stream, while a high value represents latency requirements that are easy to meet. For example, $\psi = [0.1 - 0.3]$ is difficult, since on average, $d_i^j = 0.2T$, while $\psi = [0.1 - 0.9]$ is simpler since on average, d_i^j can be up to $0.9T$.

The performance of the two methods was compared on three criteria:

- Total number of admitted streams for the message stream set. This measures the likelihood that a real-time message stream will be accepted, and is the primary concern of potential network users.
- Delay ratio distribution of admitted streams. This measures how the admission procedure admits messages with very stringent end–to–end delay requirements.
- Network utilization. This measures the degree to which network bandwidth is effectively used, and is of interest to the persons who manage a network.

The results of our experiments are presented in Tables 5.2–5.3 and Figures 5.3–5.4. For each combination of stream parameter values, 10 message stream sets were ran-

domly generated and admitted to the network. The results show the 95% confidence intervals for the performance measures obtained from these test sets.

For our first two performance criteria, the total number of admitted streams and delay ratios, the results are shown in table 5.2 and plotted in figures 5.3. The results indicate that the adaptive stream admission procedure is able to admit more streams when they have low latency bound requirements. This is apparent from the higher number of admissions by adaptive stream admission for $\psi = [0.1, 0.3]$ and [0.3,0.6]. As ψ is relaxed to $[0.6, 0.9]$, the improvement of the adaptive method over the non-adaptive method decreases.

Table 5.2: Stream admissions for selected ψ ranges

ψ	Adaptive	Non-Adaptive
	Number Admitted	
$[0.1, 0.3)$	197.5±2.36	178.9±2.43
$[0.3, 0.6)$	370.9±6.67	342.1±3.35
$[0.6, 0.9)$	371.4±5.76	371.3±5.73
	Admission Utilization	
$[0.1, 0.3)$	36.49±0.47	32.94±0.33
$[0.3, 0.6)$	64.51±1.15	61.79±0.71
$[0.6, 0.9)$	64.34±1.00	64.35±1.01

We also ran an experiment in which the range in latency requirements was quite large. This was accomplished by setting ψ to be in the range $[0.1, 0.6]$. The number of messages admitted was then tabulated according to the range of ψ used for their latencies. The results are shown in Table 5.3, and Figure 5.4. The adaptive method admitted more message streams, and in particular was much better at admitting streams with tight latency requirements.

We expect to use a real-time stream admission procedure together with a message transfer mechanism such as preemptive cut-through. Some of the message streams to be transferred with preemptive cut-through may have very tight latency requirements; having the ability to schedule these on the network is critical.

Table 5.3: Stream admissions for $\psi = [0.1, 0.6]$

ψ	Adaptive	Non-Adaptive
$[0.1, 0.2)$	52.6±3.72	34.8±3.26
$[0.2, 0.3)$	65.8±2.28	48.8±2.14
$[0.3, 0.3)$	65.2±4.46	55.9±2.27
$[0.4, 0.5)$	78.4±5.61	79.1±4.37
$[0.5, 0.6)$	80.3±4.94	95.4±6.12
Total $[0.1 - 0.6)$	342.3±5.27	314.0±4.43

Table 5.2 also illustrates the difference in link utilizations using the two methods. This experiment is for validating the type of streams admitted by adaptive stream admission. If the number of streams admitted by adaptive admissions were high but the utilization were low, this would signify that adaptive stream admission chose streams with shorter paths. The results indicate that the link utilization correlate with the number of admissions. Thus, adaptive stream admission does not choose to admit streams with shorter paths, the average path length is similar.

5.3 Discussion

This chapter proposed and evaluated an adaptive message stream admission technique. Message stream admission is a required step before messages can be transmit-

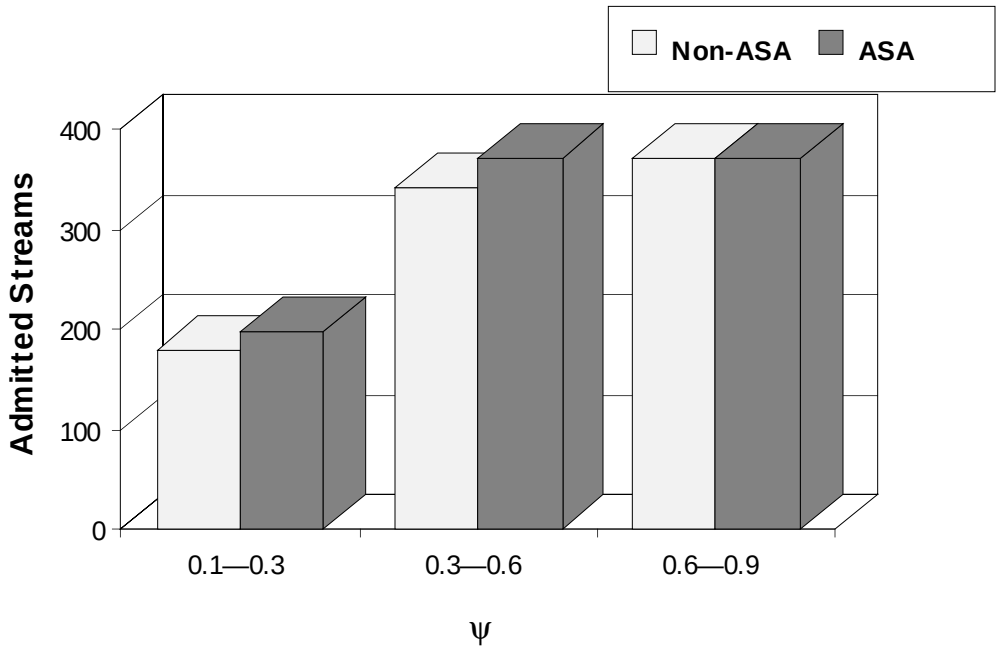


Figure 5.3: Adaptive vs. non-adaptive stream admissions

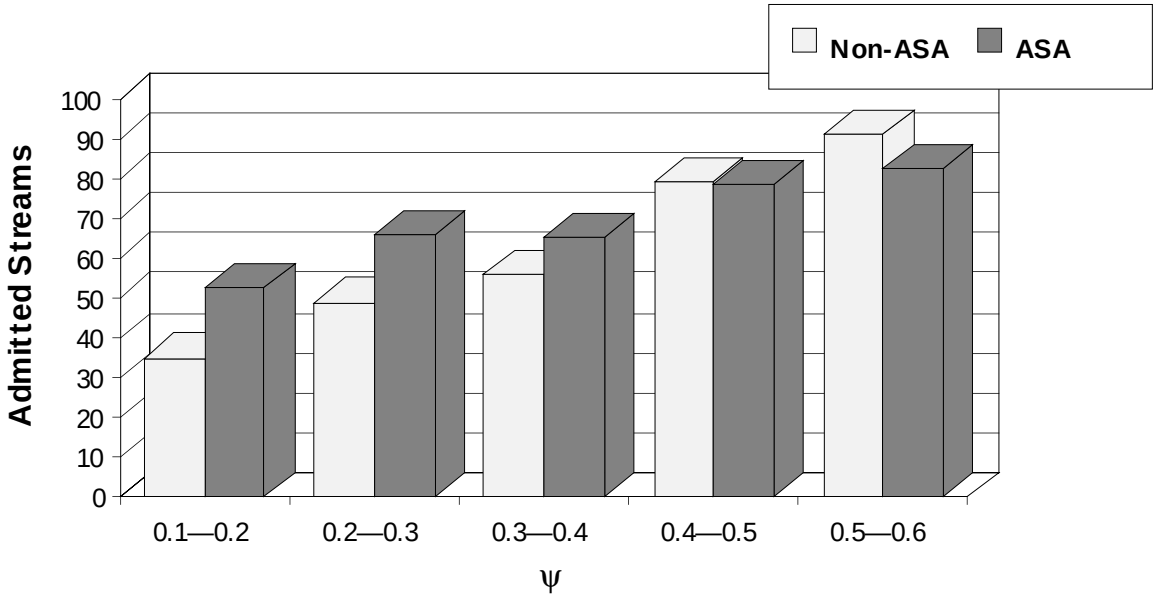


Figure 5.4: Histogram for adaptive vs. non-adaptive stream admissions—$\psi = [0.1, 0.6]$

ted in a network. The efficient allocation of network resources increases the number of streams admitted.

As far as we know, the ideas proposed here are the first study of network resource adaptation for stream admission. Our results suggest that changing link schedules by adjusting link delays allows more streams to be admitted into the network. While this is more compute intensive than simpler methods [2, 21], the computation cost is distributed over the lifetime of the message stream. On the other hand, the admission of more streams to the network means the more efficient use of network resources and a reduction in the cost of network services.

Chapter 6

Conclusions

Communication networks play an increasingly important function in many applications. In this dissertation, various aspects of communication networks at the micro and macro level are examined. At the micro level, an interconnection network for tasks with highly localized processing and dynamically changing task graphs is proposed. At the macro level, real-time communication networks for distributed real-time computing is the major focus of the research. In both studies, the objective is to improve performance by reducing communication delays.

6.1 Contributions of Research

This dissertation has made specific contributions in the areas of robotics, computer architecture, communications and real-time scheduling. The contributions are described within the context of the specific research in the following sections.

Interprocessor Communication Networks for Mobile Robot Computer Architectures

We investigated the use of a dynamically reconfigurable, bus based multiprocessor interconnection network for mobile robot computer architectures. In the robotics area, the architectural requirements in high level applications have received very little attention [47]. Our work examines the task characteristics and proposes realizable architectural enhancements.

In computer architecture, our research contributions include proposing a new connection network for multi-computers for mobile robots. We investigate the hardware implementation of this architecture at the module level and consider the technological requirements. For the operation of the segmented bus, we have both formulated a new effective transfer mechanism called preemptive circuit-switching for this architecture and also investigated the use of cut-through for these transfers. We have analyzed and simulated the performance of these data transfer mechanisms. Our models included overheads which may be encountered if the system were implemented. The results of our analysis have shown that for certain task distributions, both our proposed method and virtual cut-through were as effective as using multiple buses, but at a lower cost.

Real Time Communications for Distributed Real-Time Computing

We have approached the problem of providing guaranteed end–to–end delay in packet switched wide area networks such as B-ISDN. Such guarantees are necessary real-time data exchanges in critical applications such as military or civilian/command control systems involving remote sensors, and remote medical assistance facilities.

The problem involves two parts: The off-line quality of service guarantee and the on-line enforcement of this quality of service. We have offered two solutions to these problems. For the off-line guarantee, we have presented an adaptive stream admission procedure. Our analysis has shown that using an adaptive procedure results in the admission of more real-time message streams as compared to a non-adaptive stream admission procedure. This conclusion is especially valid when the message streams require very low end–to–end delay bounds.

The off-line guarantee check is not useful if it cannot be enforced during actual message transmission. For on-line guarantee enforcement, we have presented a preemptive cut-through message transfer mechanism. We showed that the mechanism reduces the guaranteed minimum achievable end–to–end delay to values seen in circuit-switched networks. Our technique also provides very low delay-jitter without sacrificing end–to–end delay as in previous approaches. Interactive real-time message streams are expected to use a significant portion of integrated network capacity. Provision of a quality of service at least as good as previous networks is essential to the widespread acceptance of integrated networks as a viable alternative to existing solutions.

Any performance enhancement to emerging communication standards must be achievable within the standard itself. We have examined how preemptive cut-through can be used to implement message-level delay control within an ATM switch. While current ATM switch architectures are by no means settled, a consensus seems to have emerged on the general features of the switch. We have used these features to look into the performance gains resulting from using preemptive cut-through on

ATM. Although ATM does not incorporate features for cell-level cut-through, it still benefits from using preemptive cut-through. The benefit comes in terms of reduced processing delays and reduced processor capacity requirements on each switch.

6.2 Suggestions for Future Work

Our research on communication networks for mobile robot computer architectures and distributed real-time computing has considered several basic research issues. In the course of the research, many other issues have come up, some of which deserve further study.

Segmented Bus

The segmented bus was introduced on the basis of predicted behaviors of tasks in a mobile robot. To judge the effectiveness of segmented bus experimental data on actual communication patterns and volumes are needed. The MARGE [5] project provides an excellent opportunity for this task.

In our work, we did not consider the real-time implications of the data transfer protocols on the segmented bus. The preemptive circuit switching data transfer technique is suitable for real-time data transfers on a bus. The priority mechanism can be used with static or dynamic priority real-time schedulers. While this has been examined in theory [48, 49], our work provided the first step in proposing the details of how this may be implemented. Further study in this direction may result in a mechanism that can provide a practical mechanism for real-time data transfers on a

bus based architecture.

Real-Time Communication for Distributed Real-Time Computing

For real-time communications there are several issues that could be further explored. For off-line guarantees, the adaptive stream admission procedure did not optimize the sequence of slack allocations. Even though our preliminary results using a random allocation resulted in improved performance, a smarter slack allocation should further increase the number of admitted message streams.

For on-line guarantees, the preemptive cut-through mechanism, as are most real-time scheduler based mechanisms, is very conservative in terms of resource allocation. This is due to the underlying resource allocation model which is based on process scheduling. This conservative approach can be relaxed since the underlying resource scheduling model does not fully conform to the "arrival over time" behavior of communication resources.

In practice, many protocols fail to deliver their promised performance since overheads associated with hardware result in unforeseen delays. There have been many protocols suggested for ATM. Our application of preemptive cut-through is expected to reduce some of the hardware overheads. However, to fully evaluate this reduction, detailed estimates of hardware processing times are needed. Since only two companies have even built ATM switches at this time, these estimates were not available. When they are, our results and other proposals can be compared using actual measurements.

Bibliography

[1] M. Schwartz, *Computer-Communication Network Design and Analysis.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

[2] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal of Selected Areas in Communications*, vol. 8, pp. 368–379, April 1990.

[3] H. P. Moravec, "The Stanford Cart and the CMU Rover," *Proceedings of the IEEE*, vol. 71, no. 7, pp. 872–884, 1983.

[4] S. A. Shafer, A. Stentz, and C. E. Thorpe, "An architecture for sensor fusion in a mobile robot," in *IEEE Int'l Conference on Robotics and Automation*, (San Francisco), pp. 2002–2011, 1986.

[5] R. C. Luo, C. M. Aras, H. Potlapalli, and M. H. Lin, "MARGE: Mobile autonomous robot for guidance experiments," in *SME Conference on Manufacturing and Automation*, (Pittsburgh, PA), 1991.

[6] S. Narasimhan, D. M. Siegel, and J. M. Hollerbach, "CONDOR: An architecture for controlling the Utah-MIT dextrous hand," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 616–627, October 1989.

[7] E. F. Gehringer, D. P. Siewiorek, and Z. Z. Segall, *Parallel Processing, the CM* Experience.* Digital Press, 1987.

[8] G. D. Hager, *Task-Directed Sensor Fusion and Planning: a computational approach.* Boston: Kluwer Academic Publishers, 1990.

[9] H. Li and Q. F. Stout, *Reconfigurable Massively Parallel Computers.* Englewood Cliffs, NJ: Prentice Hall, 1991.

[10] X. BaiQiang and N. Ida, "A dynamically segmented bus architecture," *Computers and Electrical Engineering*, vol. 16, no. 3, pp. 139–158, 1990.

[11] S. E. Minzer, "Broadband ISDN and asynchronous transfer mode (ATM)," *IEEE Communications Magazine*, pp. 17–24, September 1989.

[12] E. M. Gold, "The conferencing market explodes," *Networking Management*, vol. 10, pp. 40–48, May 1992.

[13] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 2 ed., 1992.

[14] T. D. C. Little and A. Ghafoor, "Multimedia synchronization protocols for broadband integrated services," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1368–1381, December 1991.

[15] J.-Y. L. Boudec, "The Asynchronous Transfer Mode: a tutorial," *Computer Networks and ISDN Systems*, vol. 24, pp. 279–309, 1992.

[16] W. R. Beam, *Command, control and communication systems engineering*. New York: McGraw-Hill, 1989.

[17] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an Integrated Services Packet Network: Architecture and mechanism," in *SIGCOMM*, 1992.

[18] P. Gopal and B. K. Kadaba, "Network delay considerations for packetized voice," *Performance Evaluation*, vol. 9, pp. 167–180, 1989.

[19] E. D. Sykas, K. M. Vlakos, and M. J. Hillyard, "Overview of ATM networks:functions and procedures," *Computer Communications*, vol. 14, pp. 615–626, December 1991.

[20] C. Dhas, V. K. Konangi, and M. Sreetharan, *Broadband Switching: Architectures, Protocols, Design and Analysis*. Los Alamitos, CA: IEEE Computer Society Press, 1991.

[21] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multihop networks," in *11th International Conference on Distributed Computing Systems*, (Arlington, TX), pp. 300–307, 1991.

[22] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point to point packet switched networks," *IEEE Transactions on Communications*, preprint 1992.

[23] K. G. Shin and M.-S. Chen, "On the number of acceptable task assignments in distributed computing systems," *IEEE Transactions on Computers*, vol. 39, pp. 99–110, January 1990.

[24] B. W. Arden and R. Ginosar, "MP/C:A multiprocessor/ computer architecture," *IEEE Transactions on Computers*, vol. C-31, pp. 455–473, May 1982.

[25] N. J. Dimopoulos, "On the structure of the homogeneous multiprocessor," *IEEE Transactions on Computers*, vol. C-34, pp. 141–150, February 1985.

[26] W. C. Athas, "Physically compact, high performance multicomputers," in *Proceedings of the Seventh MIT Conference on Advanced Research in VLSI*, (Boston, MA), pp. 302–313, 1990.

[27] T. N. Mudge, J. P. Hayes, and D. C. Winsor, "Multiple bus architectures," *Computer*, pp. 42–48, June 1987.

[28] T. Lang, M. Valero, and I. Alegre, "Bandwidth of crossbar and multiple-bus connections for multiprocessors," *IEEE Transactions on Computers*, vol. C-31, pp. 1227–1234, December 1982.

[29] C. C. Ko, K. M. Lye, K. C. Chua, and F. T. Yap, "Analysis of a CSMA/CD-based protocol with dynamic segmentation," *Computer Networks and ISDN Systems*, vol. 16, pp. 347–355, 1989.

[30] D. B. Shu and J. G. Nash, "The gated interconnection network for dynamic programming," in *Concurrent computations : Princeton Workshop on Algorithm, Architecture, and Technology Issues for Models of Concurrent Computation*, vol. 1, (New York), 1987.

[31] O. Ganz and I. Chlamtac, "Analysis of multiple-bus synchronous and asynchronous communication systems," *Performance Evaluation*, vol. 5, pp. 1–13, January 1985.

[32] W. Chu, L. J. Holloway, M.-T. Lan, and K. Efe, "Task allocation in distributed data processing," *Computer*, vol. 13, pp. 57–69, November 1980.

[33] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Transactions on Computers*, vol. 37, pp. 1384–1397, November 1988.

[34] C. E. Houstis, "Allocation of real-time applications to distributed systems," in *Proceedings of the 1987 International Conference on Parallel Processing*, (University Park, PA, USA), pp. 863–866, 1987.

[35] D. M. Nicol and J. H. Saltz, "Dynamic remapping of parallel computations with varying resource demands," *IEEE Transactions on Computers*, vol. 37, pp. 1073–1087, September 1988.

[36] F. Distante and V. Piuri, "Distributed architecture design to match optimum process allocation:a simulated annealing approach," in *Real-Time Systems Symposium*, pp. 114–123, 1987.

[37] D.-H. Du and G. Vidal-Naquet, "Mapping communicating task graphs onto reconfigurable multiprocessor architectures," in *International Symposium on Computers and Information Systems*, vol. 1, (Antalya, Turkey), 1991.

[38] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Task scheduling in distributed real-time systems," in *Proceedings - IECON '87: 1987 International Conference on Industrial Electronics, Control, and Instrumentation.*, (New York, NY, USA), pp. 909–916, 1987.

[39] A. K. Mok, P. Amerasinghe, M. Chen, S. Sutanthavibul, and K. Tantisirivat, "Synthesis of a real-time message processing system with data-driven timing constraints," in *Proceedings of the 1987 Real-Time Systems Symposium*, pp. 133–143, 1987.

[40] P. Kermani and L. Kleinrock, "Virtual cut through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[41] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–53, 1987.

[42] D. A. Reed and D. C. Grunwald, "The performance of multicomputer interconnection networks," *Computer*, pp. 63–73, June 1987.

[43] A. A. B. Pritsker, *Introduction to Simulation and SLAM II.* New York: Halsted Press, a division of John Wiley and Sons, 3 ed., 1986.

[44] W. D. Peterson, *The VMEbus Handbook.* Scottsdale, AZ, USA: VMEBus International Trade Association (VITA), 1989.

[45] M. Annaratone, *Digital CMOS Circuit Design.* Norwell, MA: Kluwer Academic Publishers, 1986.

[46] D. Hawley, "Futurebus," in *Digital Bus Handbook*, McGraw-Hill, 1990.

[47] J. S. Albus, H. G. McCain, and R. Lumia, "NASA/NBS standard reference model for telerobot control system architecture (NASREM)," Tech. Rep. 1235, National Institute for Standards and Technology, July 1987.

[48] L. Sha, R. Rajkumar, and J. Lehocszky, "Real-time scheduling support in futurebus+," in *11th Real-Time Systems Symposium*, vol. 1, (Lake Buena Visa, Florida), 1990.

[49] J. P. Lehoczky and L. Sha, "Performance of real-time bus scheduling algorithms," *Performance 86*, pp. 44–53, 1986.

[50] R. L. Cruz, "A calculus for network delay, part i: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, pp. 114–131, January 1991.

[51] S. J. Golestani, "A framing strategy for congestion management," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1064–1077, September 1991.

[52] D. P. Anderson, S.-Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews, "Support for continuous media in the DASH system," in *10th Intl. Conference on Distributed Computing Systems*, (Paris, France), pp. 54–61, 1990.

[53] D. C. Verma, H. Zhang, and D. Ferrari, "Delay jitter control for real-time communication in a packet switching network," in *Tri Comm'91 IEEE Conference on Communications Software*, (Chapel Hill, NC), pp. 35–43, 1991.

[54] A. Gersht and K. J. Lee, "A congestion control framework for ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1119–1129, September 1991.

[55] J. M. Hyman, A. A. Lazar, and G. Pacifici, "Real-time scheduling with quality of service constraints," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1052–1063, September 1991.

[56] A. A. Lazar, G. Pacifici, and J. S. White, "Real-time traffic measurements on MAGNET II," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 467–483, April 1990.

[57] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the Association of Computing Machinery*, vol. 20, pp. pp 46–61, January 1973.

[58] A. K. lau Mok, *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD thesis, Massachusetts Institute of Technology, 1983.

[59] A. M. van Tilborg and G. M. Koob, *Foundations of Real-Time Computing: Scheduling and Resource Management.* Boston/Dordrecht/London: Kluwer Academic Publishers, 1991.

[60] M. G. Hluchyj and M. J. Karol, "Queueing in high performance packet switching," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1587–1597, December 1988.

[61] D. Hong and T. Suda, "Congestion control and prevention in ATM networks," *IEEE Network Magazine*, pp. 10–19, July 1991.

[62] J. J. Bae and T. Suda, "Survey of traffic control schemes and protocols in ATM networks," *Proceedings of the IEEE*, vol. 79, pp. 170–189, February 1991.

[63] H. Ohnishi, T. Okada, and K. Noguchi, "Flow control schemes and delay/loss tradeoff in ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1609–1616, December 1988.

[64] R. K. Jurgen, "Digital video," *IEEE Spectrum*, vol. 29, pp. 24–30, March 1992.

[65] J. S. Turner, "Managing bandwidth in ATM networks with bursty traffic," *IEEE Network*, pp. 50–58, September 1992.

[66] T. D. Morris and H. G. Perros, "Approximate analysis of a discrete-time tandem network of cut-through queues with blocking and bursty traffic," in *2nd International Workshop on Queueing Networks with Blocking*, (Research Triangle Park, NC), 1992.

[67] R. Sedgewick, *Algorithms*. Reading, MA: Addison-Wesley, 1983.

[68] T.-C. Hsu and L.-Y. Kung, "A hardware mechanism for priority queue," *Computer Architecture News*, vol. 17, pp. 162–169, December 1989.

[69] Z. Fan and K. H. Cheng, "Design and analysis of simultaneous priority queues," *Journal of Parallel and Distributed Computing*, vol. 9, no. 4, pp. 387–397, 1990.

[70] R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent capacity and its application to bandwidth allocation in high speed networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 968–981, September 1991.

[71] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, November 1990.

[72] J. D. Spragins, J. L. Hammond, and K. Pawlikowski, *Telecommunications: Protocols and Design*. Reading, MA: Addison-Wesley, 1991.

Appendix A

MARGE: Mobile Autonomous Robot for Guidance Experiments

This Appendix describes the MARGE project in the Center for Robotic and Intelligent Machines (CRIM) at the North Carolina State University. While no results from MARGE can be found in this dissertation, the research and activity in MARGE is the basis of many ideas in Chapter 2 of this work.

The motivation for the MARGE project was to combine various individual projects in CRIM into a larger, integrated research activity. The project was started in December 1989 and has been headed by the author, under the supervision of Dr. Ren C. Luo, until December 1992.

The objective of MARGE is to develop an intelligent autonomous mobile robot capable of navigating in human environments, without human assistance.

MARGE Hardware

To carry out project objectives, a mobile base, also named MARGE, was built. The vehicle has vision, ultrasonic sensors, odometry sensors, contact sensitive bumpers and infra-red proximity sensors. All of these sensors are connected to a multiprocessor

computer. The first prototype vehicle was based on a TRC Labmate mobile platform from the TRC Corporation. Currently, a K2A base made by the Cybermotion Corporation is used. The current MARGE vehicle is shown in Figure A.1.

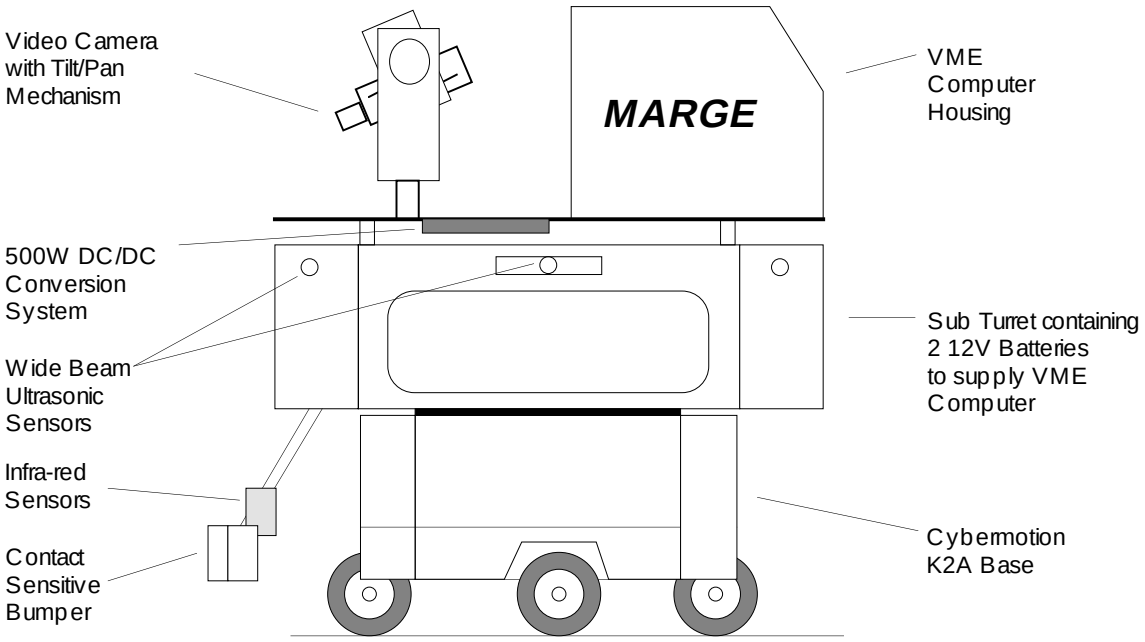


Figure A.1: The MARGE vehicle

The K2A base has a payload of about 200Kg. It has a top speed of 5m/sec. The drive mechanism is based on the synchro-drive which turns all three wheels of the base at the same speed. This provides very accurate control over the movements of the base. To turn the base, all three wheels are turned at the same time. The drive and steer motors are controlled by a Zilog Z80 based controller. The K2A base contains six ultrasonic distance sensors. These are controlled by an on-board sonar controller. Both controllers interface to the VME Bus through a single RS232 link.

The computer system on MARGE is one of the most advanced on-board multi-

processors for mobile robots (Figure A.2). This system was integrated in CRIM using components from various vendors. The computer contains four Motorola MV165 MC68040 microprocessor based boards running at 25MHz with 4 Megabytes of Random Access Memory. The processors are connected through a VME Bus. A MATRIX 40 Megabyte SCSI Hard Drive and 720 Kilobyte floppy drive are used for mass storage.

The vision system on MARGE also resides on the VME Bus. A Panasonic GP-KR402 Color video camera is attached to a DataCube Digimax Gray Scale digitizer and DataCube FrameStore unit. Three 512x512 8 bit Gray scale video frames can be stored in the FrameStore.

The Tilt and Pan mechanism mechanism, built by John Cleary of CRIM, is designed to direct the video camera and other sensors mounted on the mechanism toward objects of interest. The mechanism is controlled by a MC68HC11 microcontroller and two stepper motor controllers. The MC68HC11 is interfaced to the VME system through an RS-232 link.

Power for the VME Multiprocessor, vision system and the tilt/pan mechanism is supplied by a 500W DC/DC conversion system. The system consists of Vicor 300W 5V converter and boosters, and 100W 12V and 100W −12V converters. The DC/DC modules generate noise at frequencies of 100KHz to 3Mhz, depending on the load. To reduce this ripple noise, a capacitive π filter was made for the the 5V converter using application notes from the manufacturer. The 12 and -12V supplies use Vicor Ripple Attenuation Modules. The DC/DC converters operated off of two 12V, 65Ah Yuasa batteries connected in series. The batteries can supply the system for about

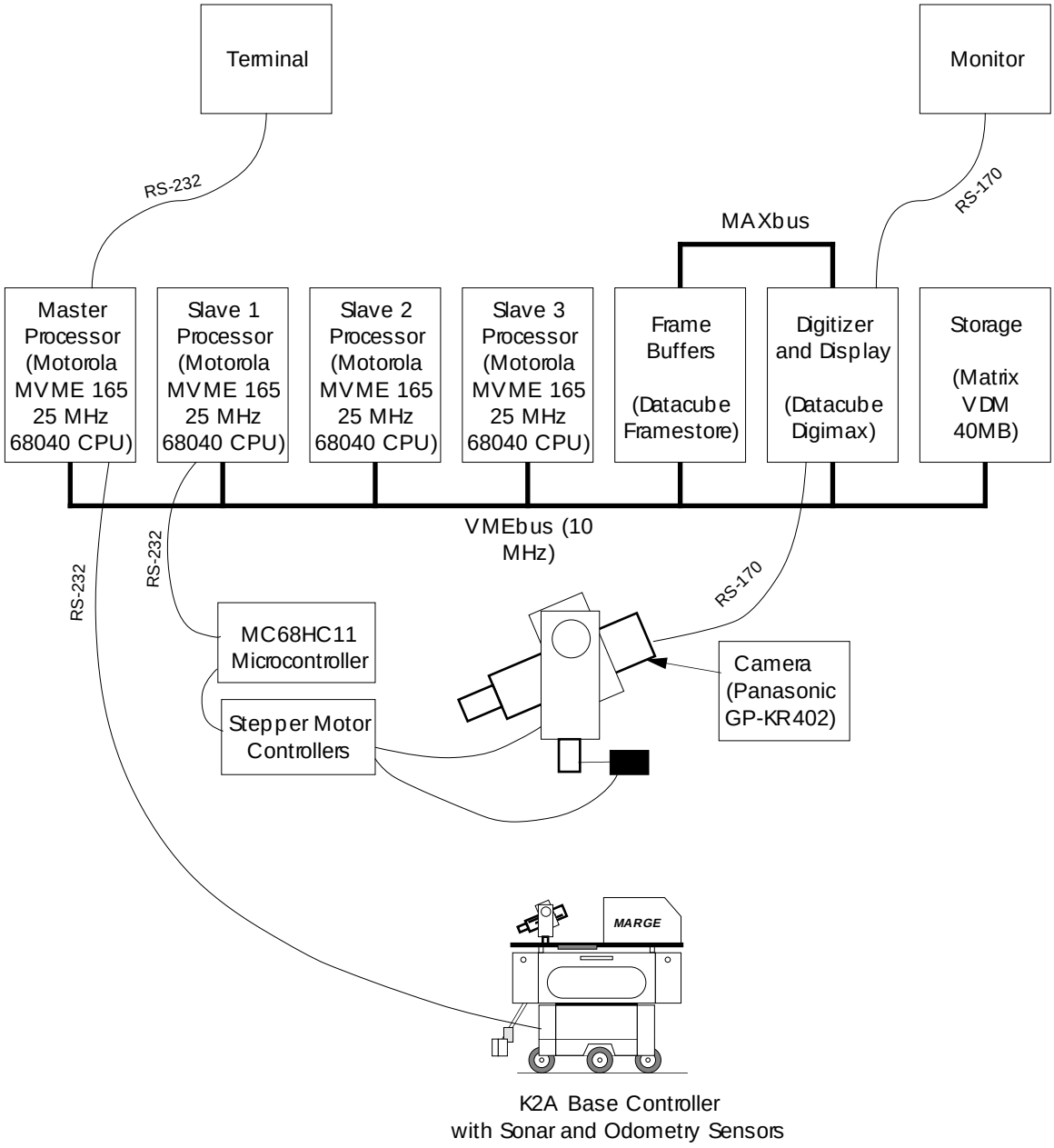


Figure A.2: The MARGE computer system

2 hours at full load. An external 30V, 30A power supply is used to both charge the batteries and supply power to the base when it is not being driven.

The distribution of power is a major problem when worst case loads approach 60 Amperes for the 5V system. Two six-gauge copper wires are used in parallel to distribute 5V power. Other lines use 10-gauge wires.

MARGE System Software

The OS/9 operating system from Microware Corporation controls the operation of the VME Multiprocessor. This operating system offers pipes, alarms, events for process synchronization, a preemptive scheduler for multi-tasking, a full file system and a flexible I/O architecture. For interprocess communications, the OS/9 pipes are used. To communication between different processors, the OS/9 pipe mechanism is extended through a software package called PNET, from Bill West Inc. of Monroe, CT.

The C language is used for all programs. The OS/9 C development environment includes a μEmacs text editor, a C compiler, assembler, optimizer, rel-code generator and a linker. The compiler includes libraries for all OS/9 calls.

The vision hardware is controlled through Maxvideo software. This software contains a OS/9 device driver, libraries and configuration tools. C and Assembly Language programs can be linked to the Maxvideo libraries to access the vision hardware.

MARGE Research Activities

The MARGE mobile robot is an invaluable tool to implement different ideas in mobile robot research. So far, vision has been the primary area of study on MARGE.

Harsh Potlapalli has used the MARGE vehicle for his research on robot landmark recognition. A Neural-Network based recognition algorithm forms the core of his effort. Dr. Michael Kay has used the vision hardware to process images for his global vision research.

Motion Planning is the second area of research on MARGE. Dr. Tai-Jee Pan has implemented the first motion planner for MARGE. A new path planner based on a geometric global database was implemented by Rachel Connors, Teebu Phillips and Bruce Ash. Jason Janet, Lee Baldwin, Barney Roberts and Andy Butler wrote the collision avoidance software for MARGE. This work has continued to include Kalman filtering to increase the accuracy and reliability of sonar measurements. Steve Goodridge's work involves reflexive motion planning using fuzzy logic. He has also developed the second generation pilot software for the MARGE vehicle.

Contributors

The MARGE project has had many contributors who have spent countless hours in the laboratory on the MARGE research and development effort. The author would like to mark these contributions. Table A.1 shows the contributors and their research in chronological order.

Other contributors from outside CRIM include Rick Hunt from MATRIX who assisted with many early problems with the VME Computer System. Bill West helped us with the obscure problems we faced during software development. The students of the School of Design at NCSU have contributed to the MARGE enclosure design effort with sketches. While their sketches were not used, they inspired our design effort and let us see how a robot is perceived outside the technical community.

Table A.1: Contributors to the MARGE project

Dr. Tai-Jee Pan	1990	Motion Planning Software
Brandon Hill Scott Woodall	1990 Spring	Sonar Sensor Microcontroller Development
Brian Allen John Cleary	1990 Summer	Sonar Sensor Microcontroller Debugging, Design and Manufacture of Sensor holders
Thomas Arredendo Tasha Helderman Kirk Krauss	1990 Fall	Sonar system development
Shrikanth Ramamurthy	1990-1991	VME Processor Software Development
Rachel Connors Teebu Phillips Bruce Ash	1991 Fall	K2A Base Collision Avoidance Software and Communication drivers
Jason Janet Lee Baldwin Barney Roberts Andy Butler	1992 Spring	K2A Collision Avoidance Software with T-Vectors and Detailed Sonar Modeling
Steven Goodridge	1992 Summer	VME Power System Upgrade
John Cleary	1992	Camera Tilt/Pan Mechanism and Controller, Enclosure Design and Manufacture
Steven Goodridge	1992 Fall	K2A Motion Control Software
Jason Janet	1992 Fall	Sonar Modeling

Harsh Potlapalli has been with MARGE since 1989. Apart from his vision research, for which he has spent a considerable amount of time, he has also arranged the purchase of the vision system and has maintained it throughout his involvement with the project.

The author has managed the MARGE team research effort. He has designed the overall system, arranged the purchase of the computer systems and has defined the hardware and software architecture currently being used on the system.