MEETING DELAY REQUIREMENTS IN COMPUTER NETWORKS WITH WORMHOLE ROUTING

By

RANIA A. BOUTROS

A Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Department of Computer Science

Faculty of Natural and Applied Sciences

Notre Dame University – Louaize

Zouk Mousbeh, Lebanon

June 1999

MEETING DELAY REQUIREMENTS IN COMPUTER NETWORKS WITH WORMHOLE ROUTING

By

Rania A. Boutros

Approved:

Fouad Chedid: Associate professor of Computer Science and Chairperson. Advisor.

Fouad Chedid

Khaldoun El Khaldi: Assistant professor of Computer Science. Member of Committee.

Nowduri Srinivas: Assistant professor of Computer Science. Member of Committee.

Jean Fares: Associate professor of Mathematics. Member of Committee.

Date of thesis defense: 2 July 99

ACKNOWLEDGEMENT

Preparing your master thesis is an endeavor that, like oil and water, doesn't mix well with having a job, a family and a marriage to prepare.

First I would like to thank God for revealing His presence in my life and for constantly expressing His love for me as I have walked through this time here at NDU.

I extend my thanks to my advisor Dr. Fouad Chedid who has been helpful in encouraging me and providing me with motivational insights throughout my studies and research.

I am thankful for the love and support expressed by my parents, sisters, and my fiancé. Their constant support has carried me through some very difficult moments.

I also appreciate the help I have received from Tony Wakim, Shady Shammas, Michel Kokosaki, and Emile Mansour who have given of themselves and their time to help me in achieving my work.

ABSTRACT

A well known problem with wormhole-routed packet networks is the potentially large amount of blocking that packets can experience due to link contention. Because of the very limited amount of buffering in such networks, blocked packets remain in the network and keep using network resources. Thus, blocked packets may in turn cause other packets to be blocked. This may affect a large number of packets over a large portion of the network. Proper connection management strategies and appropriate protocols must be devised to ensure that blocking of packets due to link contention is bounded. In [3], Zhao et. al. have developed a transmission control scheme that regulates the rate of transmission at each source node. A worst-case achievable utilization of 50% could be proved, using a simple regulated admission control scheme. The work of [3] assumes message streams of fixed-sized messages and fixed inter-arrival periods.

In this thesis, we review the problem of routing in computer networks. Then, we examine how well does the work of [3] generalizes to message streams of variable length messages and variable inter-arrival periods.

Our simulation shows that the algorithms present in [3] can be generalized even to a stochastic network. Comparisons are presented to show the difference in performance between the regulated and unregulated methods.

TABLE OF CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTERS

I. DEFINING THE PROBLEM 1

II- PARALLEL PROCESSING 3
2.1 HISTORICAL REVIEW 3
2.2 PARALLEL COMPUTATION 5
2.3 INTERCONNECTION NETWORKS 9

III- ROUTING ALGORITHMS 14
3.1 DESIGN GOALS 14
3.2 METRICS 15
3.3 EXAMPLES OF
ROUTING ALGORITHMS 16
3.4 STORE AND FORWARD
VERSUS WORMHOLE ROUTING 18
3.5 MEETING DELAY REQUIRMENTS
IN REAL TIME APPLICATIONS 24

IV- MEETING DELAY REQUIREMENTS
WITH WORMHOLE ROUTING 28
4.1 INTEL PARAGON SYSTEM 28
4.2 MESSAGE MODEL 30
4.3 TRANSMISISON CONTROL METHODS 31
4.4 ALGORITHM A1 AND A2 APPLIED TO
VARIABLE C_i AND P_i 35

V- SIMULATION RESULTS 40

VI- CONCLUSION AND FUTURE RESEARCH 42

APPENDIX: THE CODE .....

REFERENCES .....

LIST OF TABLES

Table

1. Interconnection Network parameters 13

2. A1 Routing efficiency function of C_i & P_i 40

3. A2 Routing efficiency function of C_i & P_I 41

LIST OF FIGURES

Figure

1. UMA multiprocessor model 6
2. NUMA multiprocessor model 7
3. COMA model for multiprocessor 8
4. Message-passing multicomputer model 9
5. Linear Array 11
6. Ring 11
7. Mesh 12
8. 3-cube 12
9. 4-cube 12
10. E-cube routing on a hypercube computer 17
11. X-Y routing on a 2D-Mesh computer 17
12. Store and Forward routing 18
13. Asynchronous pipeline model 19
14. Asynchronous model using handshaking protocol 20
15. Synchronous pipeline model 21
16. Wormhole routing 22
17. Latency for Store and Forward routing 23
18. Latency for Wormhole routing 24
19. Intel Paragon node architecture 29
20. Intel Paragon router architecture 29
21. Traditional transmission control method 33
22. A1 running algorithm .....
23. A2 running algorithm

24. A1 Routing efficiency function of C_i & P_i.....

25. A2 Routing efficiency function of C_i & P_I.....

CHAPTER I

Defining the Problem

The increased development and use of networked real-time applications in distributed multimedia, remote laboratoring, and distributed virtual reality has generated a large amount of interest in the development of real-time communication protocols to support such applications. Such protocols provide real-time guarantees for message deliveries to the network clients, typically in terms of bounded delay and jitter. This in turn enables the applications to meet their end-to-end time requirements. In order to provide such guarantees on message delivery, resources in the network must be appropriately allocated. The admission control component of the protocol must ensure that enough resources are available in the system so that new connections can in fact be guaranteed the required performance.

Due to the increasing bandwidth requirements in a large number of applications such as incorporation of uncompressed video streams in control loops for remote laboratoring, a number of very-high speed networking technologies are currently under investigation for their applicability to support hard-real-time communication, such as ATM, FDDI, GPPI. As exemplified by recently developed technology, wormhole-routed networks are a promising approach for high-bandwidth, low-latency communication for small and medium-sized networks.

A well known problem with wormhole-routed packet networks is the potentially large amount of blocking that packets can experience of link contention. Because of the very limited amount of buffering in such networks, blocked packets remain in the network and keep using network resources. Thus, they may in turn

cause other packets to be blocked. This may affect a large number of packets over a large portion of the network. Proper connection management strategies and appropriate protocols must be devised to ensure that blocking of packets due to link contention is bounded.

The rest of this thesis is organized as follows: chapter II introduces Parallel processing, chapter III introduces the Routing problem and reviews the Store and Forward and the Wormhole routing networks, chapter IV introduces the work done in [3], Chapter V includes our simulation results, and chapter V concludes with the conclusion and ideas for future research.

CHAPTER II

Parallel Processing

Parallel processing has emerged as a key enabling technology in modern computers, driven by the ever-increasing demand for higher performance, lower costs, and sustained productivity in real-life applications. Concurrent events are taking place in today's high performance computers due to the common practice of multiprogramming, multiprocessing, and multicomputing.

2.1 Historical Review

Over the past five decades, electronic computers have gone through five generations of development. First-generation computers were built with a single Central Processing Unit (CPU) which performed serial fixed-point arithmetic using a program counter, branch instructions, and an accumulator machine or assembly languages were used.

Index registers, floating-point arithmetic, multiplexed memory, and I/O processors were introduced with second-generation computers. High-level languages (HLLs), such as Fortran, ALGOL, and Cobol, were introduced along with compilers, subroutine libraries, and batch processing monitors.

Microprogrammed control became popular with the third generation. Pipelining and cache memory were introduced to close up the speed gap between the CPU and main memory. The idea of multiprogramming was implemented to interleave CPU and I/O activities across multiple user programs. This led to the

development of time-sharing operating systems (OS) using virtual memory with greater sharing or multiplexing of resources.

Parallel computers in various architectures appeared in the fourth generation of computers using shared or distributed memory or optional vector hardware. Multiprocessing OS, special languages, and compilers were developed for parallelism. Software tools and environments were created for parallel processing or distributed computing.

In the fifth generation, machines emphasize massively parallel processing (MPP). Scalable and latency-tolerant architectures became a must in MPP systems.

The computing industry has grown up with promises of doubling processing power and halving equipment costs every 18 months or so. That pace hasn't slowed. We have vastly more powerful hardware today than we had even three years ago.
But the demand of real-time applications is increasing the need for processing power at an even faster rate. Systems, nowadays, should provide the maximum performance in order to work with real-time applications, and we should look to open new avenues for gaining performance, even as microprocessors continue to advance. Unfortunately, almost all existing systems were originally designed decades ago, when the idea of personal computers dealing with real-time video, audio, communications and other bandwidth applications was practically science fiction.
As the importance of audio, video and interactive communications has increased in recent years, devising increasingly clever and complex methods of delivering the performance required of media based applications is a necessity because the foundations of today's systems were simply not designed with high-bandwidth media in mind.

The goal of routing for supporting real-time applications involving audio/video traffic should be computing paths that satisfy the given Quality of Service (QoS) requirements of the applications while managing the network resources efficiently.

Supporting real-time applications in high-speed networks requires reservation of resources. Since network resources are limited, efficient routing strategy and admission control are needed.

2.2 Parallel Computation

There are two major classes of parallel computers, namely shared-memory multiprocessors and message-passing multicomputers. The major distinction between multiprocessors and multicomputers lies in memory sharing and the mechanisms used for interprocessor communication. The processors in a multiprocessor system communicate with each other through shared variables in a common memory. Each computer node in a multicomputer system has a local memory, unshared with other nodes. Interprocessor communication is done through message passing among the nodes.

2.2.1 Shared-Memory Multiprocessors:

Below are described three shared-memory multiprocessor models: the uniform-memory-access (UMA) model, the nonuniform-memory-access (NUMA) model, and the cache-only memory architecture (COMA) model.

A. UMA Model

In a UMA multiprocessor model (figure .1), the physical memory is uniformly shared by all the processors. All processors have equal access time to all memory

words, which is why it is called uniform memory access. Each processor may use a private cache. Peripherals are also shared in some fashion. Multiprocessors are called tightly coupled systems due to the high degree of resource sharing. The system interconnect takes the form of a common bus, a crossbar switch, or a multistage network.

When all processors have equal access to all peripheral devices, the system is called a symmetric multiprocessor. In this case, all the processors are equally capable of running the executive programs such as the OS kernel and I/O service routines.
In an asymmetric multiprocessor, only one or a subset of processors are executive capable. An executive or a master processor can execute the operating system and handle I/O. The remaining processors have no I/O capability and thus are called attached processors.

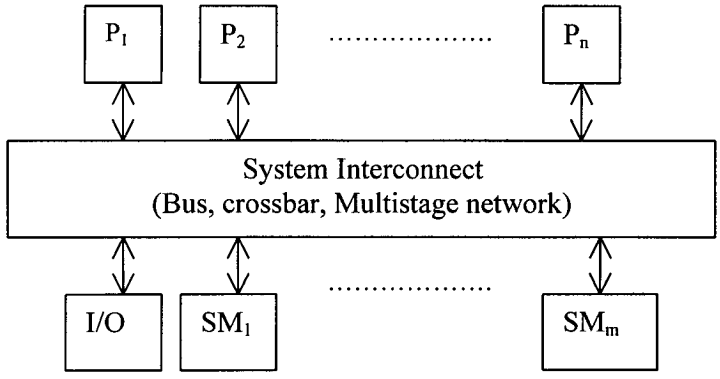


Figure 1: UMA multiprocessor model

B. NUMA Model

A NUMA multiprocessor is a shared-memory system in which the access time varies with the location of the memory word (figure 2). The shared memory is physically distributed to all processors, called local memories. The collection of all local memories forms a global address space accessible by all processors. It is faster

to access a local memory with a local processor. The access of remote memory attached to other processors takes longer due to the added delay through the interconnection network.

Besides distributed memories, globally shared memory can be added to a multiprocessor system. In this case, there are three memory-access patterns: the fastest is local memory access. The next is global memory access. The slowest is access of remote memory.

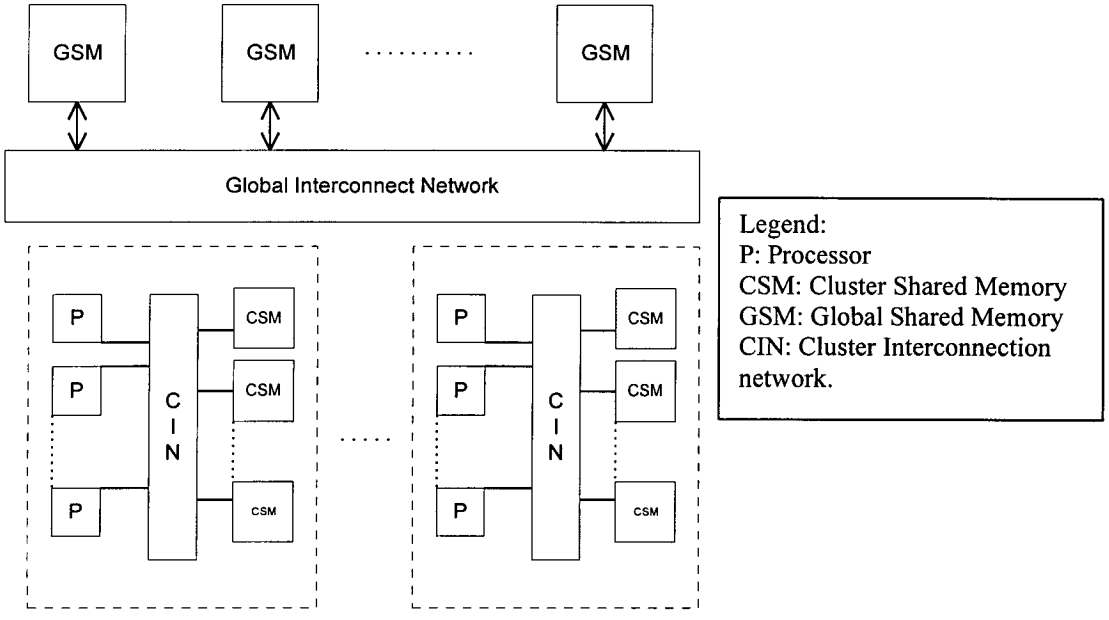


Figure 2: NUMA multiprocessor model

C. COMA Model

A multiprocessor using cache-only memory assumes the COMA model (figure .3). The COMA model is a special case of a NUMA machine, in which the distributed main memories are converted to caches. There is no memory hierarchy at each processor node. All the caches form a global address space. Remote cache access is assisted by the distributed cache directories.

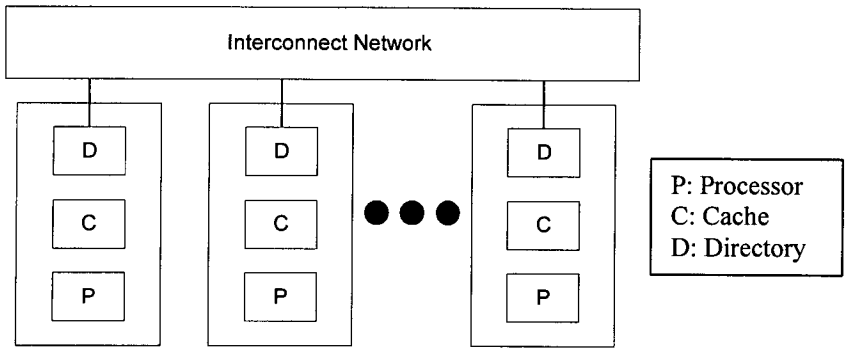


Figure 3: COMA model of a multiprocessor

Multiprocessor systems are suitable for general-purpose multiuser applications where programmability is the major concern. A major shortcoming of multiprocessors is the lack of scalability. It is rather difficult to build MPP machines using centralized shared-memory model. Latency tolerance for remote memory access is also a major limitation.

2.2.2 Distributed Memory Multicomputers

A distributed-memory multicomputer system is shown in figure 4.

The system consists of multiple computers, often called nodes, interconnected by a message-passing network. Each node is an autonomous computer consisting of a processor, local memory, and sometimes attached disks or I/O peripherals.

The message-passing network provides point-to-point static connections among the nodes. All local memories are private and are accessible only by local processors. For this reason, traditional multicomputers have been called no-remote-memory-access (NORMA) machines. However, this restriction is removed in multicomputers with distributed shared memories. Internode communication is carried out by passing messages through the static connection network.

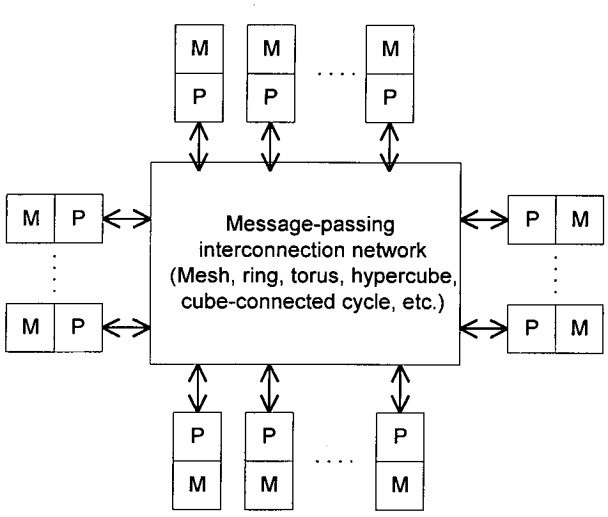


Figure 4: Message-passing multicomputer model

Message-passing multicomputers have gone through three generations of development.

The First generation (1983-1987) was based on processor board technology using hypercube architecture and software-controlled message switching. The Caltech Cosmic and Intel iPSC/1 represented the first generation development.

The second generation (1988-1992) was implemented with mesh-connected architecture, hardware message routing, and a software environment for medium-grain distributed computing, as represented by the Intel Paragon.

The third generation (1993-1997) is fine-grain multicomputers, like MIT J-machine.

2.3 Interconnection Networks

In general, a network is represented by the graph of a finite number of nodes linked by directed or undirected edges. So let us define several parameters often used to estimate the complexity, communication efficiency, and cost of a network.

The number of nodes in the graph is called the network size.

The number of edges (links or channels) incident on a node is called node degree d. The node degree reflects the number of I/O ports required per node, and thus the cost of a node. Therefore, the node degree should be kept a constant, as small as possible to reduce cost.

The diameter D of a network is the maximum shortest path between any 2 nodes. The path length is measured by the number of links traversed. The network diameter should be as small as possible from a communication point of view.

When a given network is cut into two equal halves, the minimum number of edges along the cut is called the channel bisection width b. The bisection width provides a good indicator of the maximum communication bandwidth along the bisection of a network.

Another quantitative parameter is the wire length between nodes. We say a network is symmetric if the topology is the same looking from any node.

The topology of an interconnection network can be either static or dynamic.

Static networks are formed of point-to-point direct connections which will not change during program execution.

Dynamic networks are implemented with switched channels, which are dynamically configured to match the communication demand in user programs.

Static networks are used for fixed connections among subsystems of a centralized system or multiple computing nodes of a distributed system. Dynamic networks include buses, crossbar and multistage networks, which are often used in shared-memory multiprocessor.

Static networks use direct links which are fixed once built. We describe some of their topologies below in terms of network parameters.

- Linear Array: This is a 1-dimensional network in which N nodes are connected by N-1 links in a line (figure 5). Internal nodes have degree 2 and the terminal nodes have degree 1. The diameter is N-1, which is rather long for N. The bisection width b=1. The structure is not symmetric. For very small N, say N=2, it is rather economic to implement a linear array. As the diameter increases linearly with respect to N, it should not be used for large N.

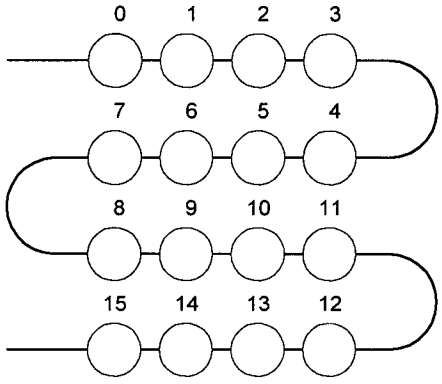


Figure 5: Linear Array

- Ring: A ring is obtained by connecting the two terminal nodes of a linear array with one extra link (figure 6). It is symmetric with a constant node degree of 2, the diameter is N/2 for a bidirectional and N for unidirectional.

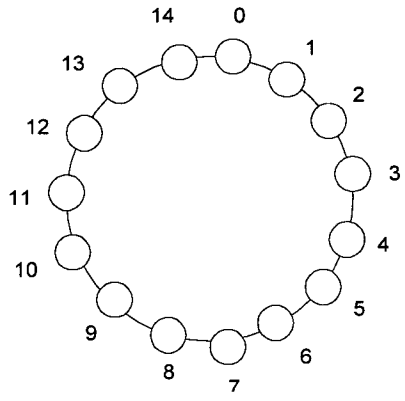


Figure 6: Ring

- Mesh: A 3x3-mesh network is show in figure 7. This is a popular architecture which has been implemented in the Intel Paragon. In general, a k-dimensional mesh with $N = n^k$ nodes has an interior node degree of 2k and the network diameter is k(n-1).

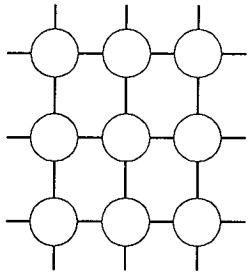


Figure 7: Mesh

- Hypercube: This is a binary n-cube architecture. In general, an n-cube consists of $N=2^n$ nodes spanning along n dimensions, with 2 nodes per dimension.

A 3-cube with 8 nodes is shown in figure 8.

A 4-cube can be found by interconnecting the corresponding nodes of two 3-cubes as shown in figure 9.

The node degree of an n-cube equals n and the network diameter also. The node degree increases linearly with respect to the dimension.

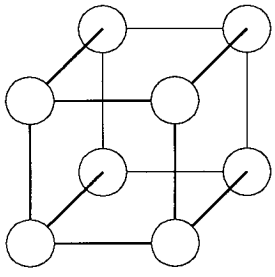


Figure 8: 3-cube

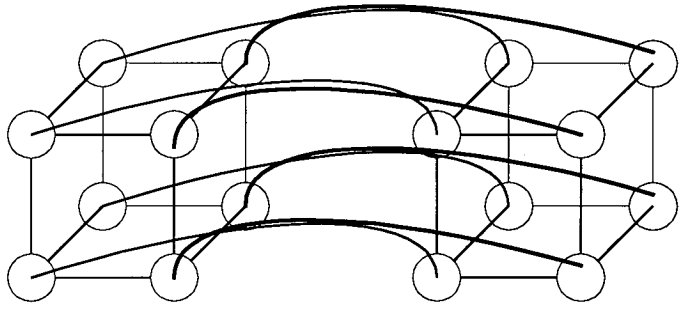


Figure 9: 4-cube

Table 1 summarizes the parameters of the different architectures mentioned above.

Network type	Node degree, d	Network diameter, D	No. of links, l	Bisection width, B	Symmetry
Linear Array	2	N-1	N-1	1	No
Ring	2	N/2	N	2	Yes
2D-Mesh	4	2(r-1)	2N-2r	R	No
Hypercube	N	N	Nn/2	N/2	Yes

Table 1: Interconnection Network parameters

The versality of a routing network will reduce the time needed for data exchange and thus can significantly improve the system performance.

CHAPTER III

Routing Algorithms

Routing is moving information across an internetwork from source to destination. Along the way, at least one intermediate node is typically encountered. Routing algorithms can be differentiated based on several key characteristics. First, the particular goals of the algorithm designer affect the operation of the resulting routing protocol. Second, there are various types of routing algorithms. Each algorithm has a different impact on network resources. Finally, routing algorithms use a variety of metrics that affects calculations of optimal routes.

3.1. Design Goals

Routing algorithms often have one or more of the following design goals:

- Optimality: It refers to the ability of routing algorithm to select the "best" route which depends on the metric and metric weightings used to make the calculation.
- Simplicity: Routing algorithms are also designed to be as simple as possible. That is, they must offer their functionality efficiently with a minimum of software and utilization overhead.
- Robustness: Routing algorithms should perform correctly in the face of unusual or unforeseen circumstances such as hardware failures, high load conditions, and incorrect implementation. Because routers are located at network junction points, they can cause considerable problems when they fail.
- Rapid convergence: Convergence is the process of agreement, by all routers, on optimal routes. When a network event causes routes to either go down or become

available, routers distribute routing update messages. Routing algorithms that converge slowly can cause routing loops or network outages.

- Flexibility: routing algorithms should quickly and accurately adapt to a variety of network circumstances. Routing algorithms can be programmed to adapt to changes in network bandwidth, router queue size, network delay, and other variables.

3. 2 Metric

Routing algorithms have used many different metrics to determine the best route. Sophisticated routing algorithms can base route selection on multiple metrics, combining them in a single (hybrid) metric. All of the following metrics have been used:

- Path length: Some routing protocols allow network administrators to assign arbitrary costs to each network link. In this case, path length is the sum of the costs associated with each link traversed. Other routing protocols define hop count, a metric that specifies the number of passes through internetworking (such as routers) that a packet must take en route from a source to a destination.
- Delay: Routing delay refers to the length of time required to move a packet from source destination through the internetwork. Delay depends on many factors, including the bandwidth of intermediate network links, the port queues at each router along the way, network congestion on all intermediate network links. Because it is a combination of several important variables, delay in a common and useful metric.
- Bandwidth: It refers to the available traffic capacity of a link. Although bandwidth is a rating of the maximum attainable throughput on a link, routes

through links with greater bandwidth do not necessarily provide better routes than routes through slower links.

- Load: It refers to the degree to which a network resource (router) is busy. It can be calculated in a variety of ways, including CPU utilization and packets processed per second.

3.3 Examples of Routing Algorithms

In the following, we describe two routing algorithms: the E-cube used with hypercubes and the X-Y used on a 2D-Mesh.

- E-cube Routing algorithm: Consider an n-cube with $N=2^n$ nodes. The source node is $s = s_{n-1} \ldots s_1 s_0$ and the destination node is $d = d_{n-1} \ldots d_1 d_0$. We want to determine a route from s to d with a minimum number of steps. We denote the n dimensions as i=1,2,…,n where the i^{th} dimension corresponds to the (i-1) bit in the node address. Let $v = v_{n-1} \ldots v_1 v_0$ be any node along the route. The route is determined as follows:

1. Compute the direction bit $r_i = s_{i-1} \oplus d_{i-1}$ for all n dimensions (i = 1,…,n).

Start the following with dimension i =1 and v = s.

2. Route from the current node v to the next node $v \oplus 2^{i-1}$ if $r_i = 1$. Skip this step if $r_i = 0$.

3. Move to dimension i+1 (i.e., i← i+1) if i[n go to step 2, else quit.

Example: n=4, s=0110, d=1101. So, $r=r_4r_3r_2r_1=1011$.

Route from s to $s\oplus 2^0 = 0111$ since r1 = 0⊕1 =1. Route from v = 0111 to $v\oplus 2^1 = 0101$ since $r_2 = 1\oplus 0=1$. Skip dimension i=3 because $r^3 = 1\oplus 1 =0$. Route from v=0101 to $v\oplus 2^3 =1101= d$ since $r^4=1$. Note that if the i^{th} bit of s and d agree, no

routing is needed along dimension i. Otherwise move from the current node to the other node along the same dimension. The procedure is repeated until the destination is reached. (Figure 10).

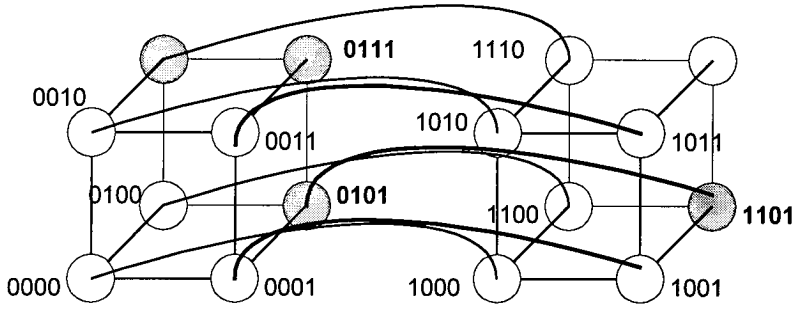
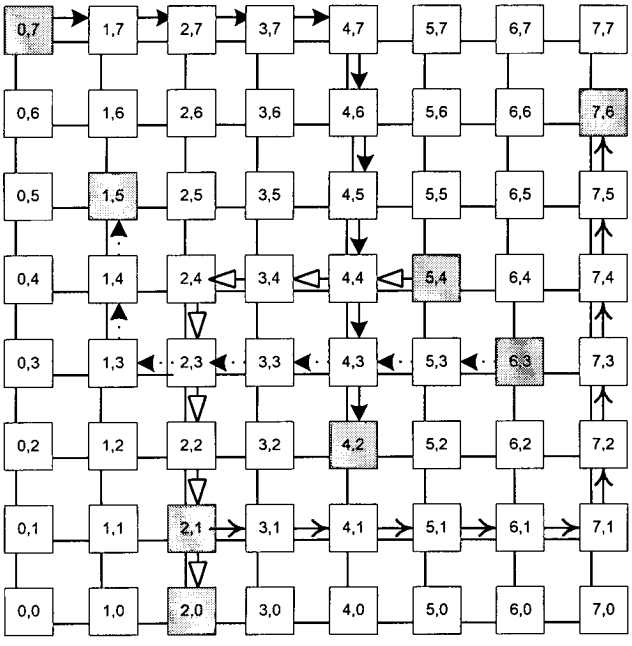


Figure 10: E-cube routing on a hypercube computer

- X-Y Routing: From any source node s=(x1,y1) to any destination node d=(x2,y2), route from s along the x-axis first until it reaches the column y2, where d is located. Then route to d along the y-axis. There are four possible X-Y routing patterns corresponding to the east-north, east-south, west-north, and west-south paths chosen.

  Example:



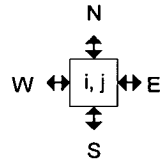


Figure 11: X-Y routing on a 2D-mesh computer

An east-north route is needed from node (2,1) to node (7,6). An east-south is set up from node (0,7) to node (4,2). A west-south route is needed from node (5,4) to (2,0). The fourth route is west-north bound from node (6,3) to node (1,5).

The X-dimension is always routed first and then the Y-dimension; a deadlock or circular wait situation will not exist.

3. 4 Store and Forward Versus Wormhole Routing

3.4.1 Store and Forward Routing

In multicomputers, with Store and Forward routing, packets are the smallest unit of information transmission. The concept is illustrated in figure12

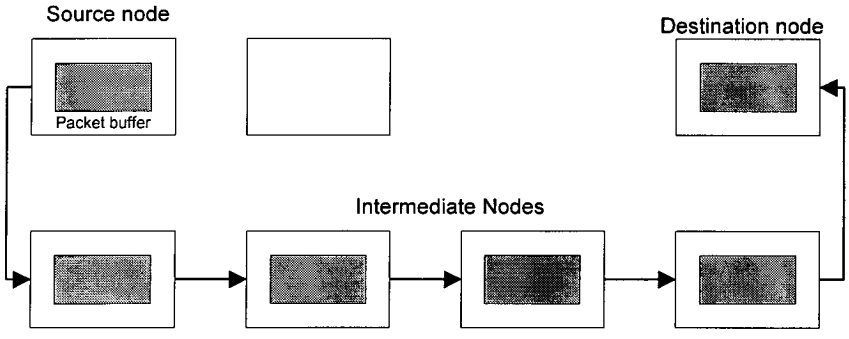


Figure 12: Store and Forward routing

Each node is required to use a packet buffer. A packet is transmitted from a source node to a destination node through a sequence of intermediate nodes.

When a packet reaches an intermediate node, it is first stored in the buffer. Then it is forwarded to the next node if the desired output channel and a packet buffer in the receiving node are both available.

The latency in store-and-forward networks is directly proportional to the distance (the number of hops) between the source and destination.

3.4.2 Pipelining

A linear pipeline processor is constructed with *k* processing stages. External inputs are fed into the pipeline at the first stage *S1*. The processed results are passed from stage S_i to stage S_{i+1} for all i= 1,2,...,k-1. The final result emerges from the pipeline at the last stage S_k.

Depending on the control of data flow along the pipeline, we model linear pipelines in two categories: *asynchronous* and *synchronous*.

A. Asynchronous Model

As shown in figure 13, data flow between adjacent stages in an asynchronous pipeline is controlled by a handshaking protocol. When stage S_i is ready to transmit data, it returns an *acknowledge* signal to S_i.

Asynchronous pipelines are useful in designing communication channels in message-passing multicomputers where pipelined wormhole routing is practiced. Asynchronous pipelines may have a variable throughput rate. Different amounts of delay may be experienced in different stages.

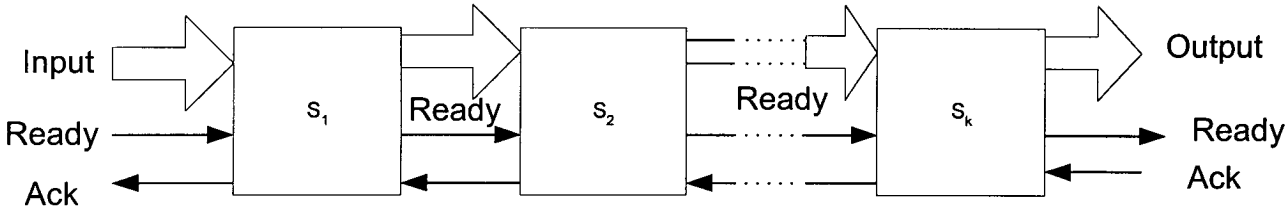


Figure 13: Asynchronous pipeline model

The pipelining of successive flits in a packet is done asynchronously using a handshaking protocol as shown in figure 14

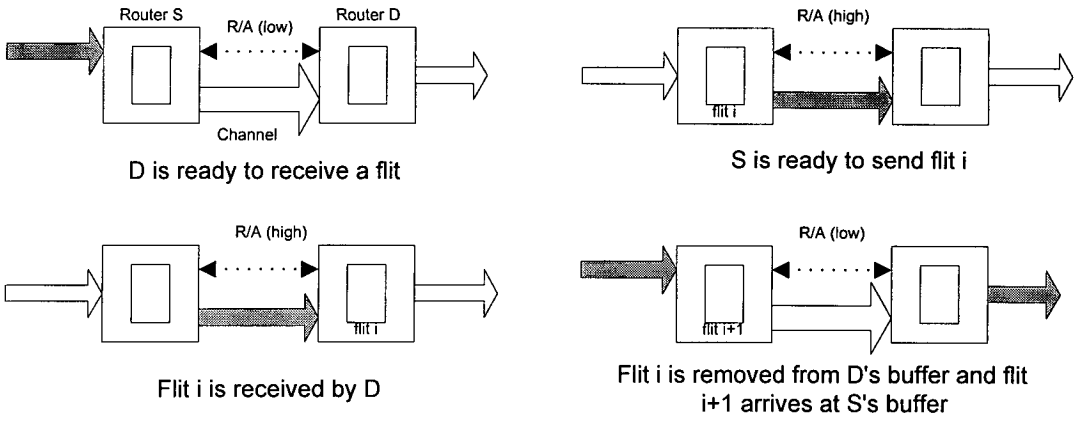


Figure 14: Asynchronous model using handshaking protocol

Along the path, a 1-bit *ready/request* (R/a) line is used between adjacent routers. When the receiving router (D) is ready to receive a flit (the flit buffer is available) it pulls the R/A line low. When the sending router (S) is ready, it raises the line high and transmits flits I through the channel. While the flit is being received by D, the R/A line is kept high. After flit *i* is removed from D's buffer (it is transmitted to the next node), the cycle repeats itself for the transmission of the next flit I+1 until the entire packet is received. Asynchronous pipelining can be very efficient, and the clock used can be faster than that used in a synchronous pipeline. However, the pipeline can be stalled if flit buffers or successive channels along the path are not available during certain cycles. Should that happen, the packet could be buffered, blocked, dragged, or detoured.

B. Synchronous Model

Synchronous pipelines are illustrated in figure 15. Clocked latches are used to interface between stages. The latches are made with master-slave flip-flops, which can isolate inputs from outputs. Upon the arrival of a clock pulse, all latches transfer data to the next stage simultaneously.

The pipeline stages are combinational logic circuits. It is desired to have approximately equal delays in all stages. These delays determine the clock period and thus the speed of the pipeline.

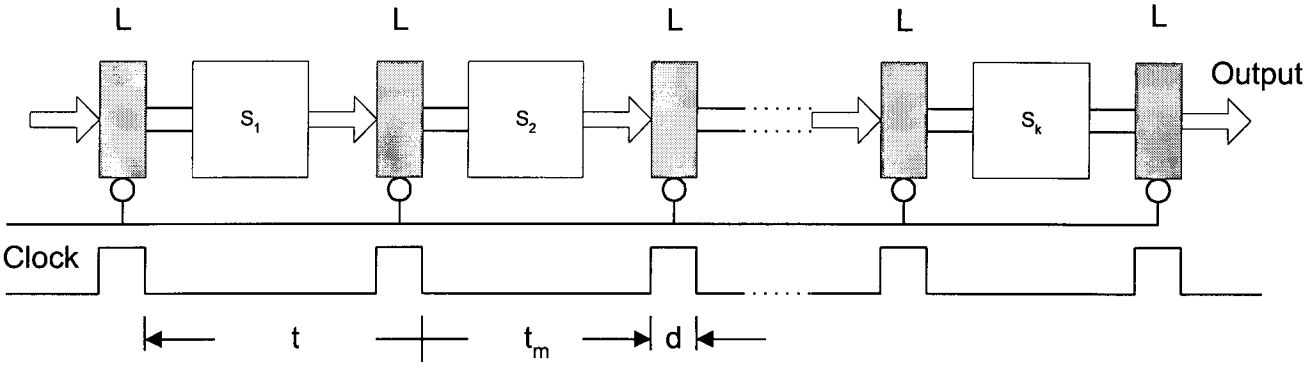


Figure 15: A synchronous pipeline model

3.4.3 Wormhole Routing

A common interconnection network switching technology that is used in MPP's such as the Intel Paragon is wormhole routing, a switching technique whereby packets are immediately forwarded to the proper output port upon arrival.

In a wormhole routing, a message is transmitted as a contiguous sequence of flits (flow control units) and the sequence of moves along the path from the source to the destination in a pipelined manner. There are two defining characteristics of wormhole routing:

- Message contiguity: every edge along the path must transmit all flits of the message in a contiguous manner, i.e. the bits of two different messages cannot be interleaved.
- Minimal buffering: each intermediate node can only buffer few flits. Hence, if the head of the message cannot move forward because another message is using the edge it wants, then the message must wait, and as it waits, it occupies a contiguous sequence of edges along its path.

Wormhole routing is a cost-effective way to provide very-high speed, very-low latency communication for emerging distributed real-time applications with high-bandwidth demands. As long as the worm is using this output port, no other worm can use it. The output port is released when the tail of the worm passes through the switches along the path of the worm.

Unfortunately, the lack of traffic buffering in the switches makes it difficult to give real-time guarantees to traffic streams that contend for communication links.

By subdividing the packet into smaller flits, newer multicomputers implement the wormhole routing scheme shown in figure 16.

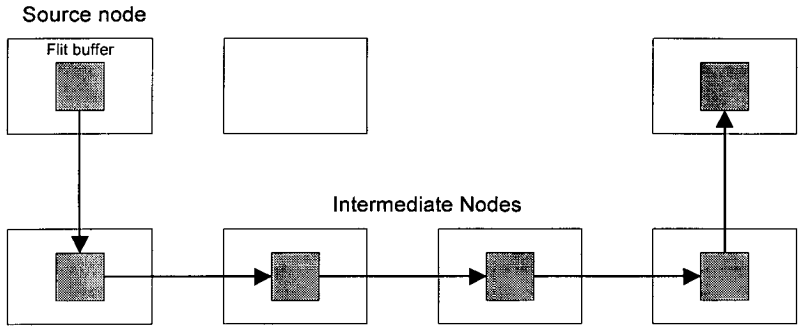


Figure 16: Wormhole routing

All the flits in the same packet are transmitted in order as inseparable companions in a pipelined fashion. The packet can be visualized as a railroad train with an engine car (the header flit) towing a long sequence of box cars (data flits). Only the header flit knows where the train (packet) is going. All the data flits must follow the header flit. Different packets can be interleaved during transmission. However, the flits from different packets cannot be mixed up. Otherwise, they may be towed to the wrong destinations. Wormhole routing has latency almost independent of the distance between the source and destination.

3.4.4 Latency Comparison Between Store-and-Forward and Wormhole Routing

Let L be the packet length (in bits), W the channel bandwidth (in bits/s), D the distance (number of nodes traversed minus 1) and F the flit length (in bits).

The communication latency T_{SF} for a store-and-forward network is expressed by:

$$T_{SF} = \frac{L}{W}(D+1)$$

T_{SF} is directly proportional to D

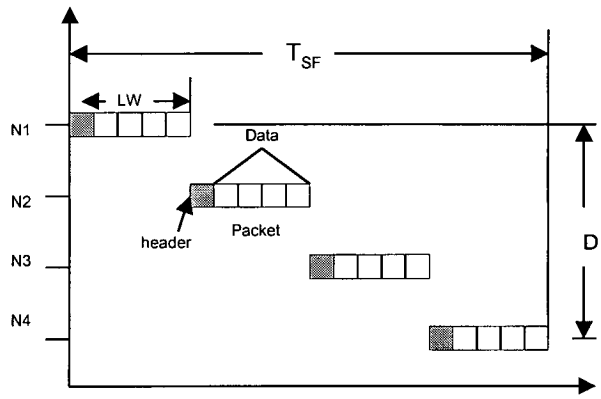


Figure 17: Latency for Store and Forward routing.

The communication latency for a wormhole routed network is:

$$T_{WH} = \frac{L}{W} + \frac{F}{W} * D$$

T_{WH}= L/W if L>>F. thus the distance D has a negligible effect on the routing latency.

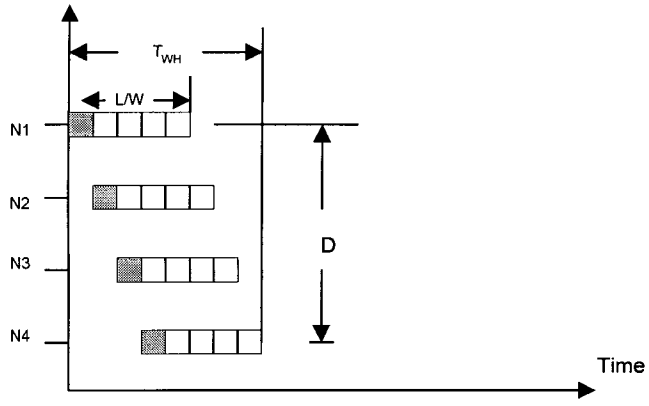


Figure 18: Latency for Wormhole routing

3.5 Meeting Delay Requirements in Real Applications

In a real-time system, timing constraints must be met for the application to be correct. In a distributed real-time system, many of these time constraints are end-to-end, and often require the scheduling of different resources (e.g., processors on each node and the communication facilities between them).

The primary performance metric for real-time computing is acceptable predictability of the timeliness of a set of activities. This applies to both hard and soft real-time cases. In contrast, non real-time computing performance is usually measured in terms of throughput.

One of the things that makes real-time resource management so much more difficult than non real-time resource management is that, the real-time performance requirement of acceptable predictability of timeliness must be met along with other

requirements, such as precedence constraints and resource utilization conflict resolution constraints.

Real-time systems require that timing constraints be expressed, enforced, and their violations handled.

Timing constraints may be expressed as absolute time or relative time. Absolute time may be based on wall clock time or a system global time. Absolute time may be served centrally or in a decentralized manner; the latter usually requires some form of clock synchronization. Relative time is relative to when the request is made.

The following terms are often used to express timing constraints:

- An "earliest start time" constraint specifies a time before which the activity may not start. That is, the task must wait for the specified time before it may start.
- A "latest start time" constraint specifies a time before which the activity must start. That is, if the activity has not started by the specified time, an error has occurred. Latest start times are useful to detect potential violations of planned schedules or eventual deadline violations before they actually occur.
- A "deadline" specifies a time before which the activity must complete.
- A periodic constraint specifies earliest start times and deadlines at regular time intervals for repeated instances of an activity.

The delay experienced by the service traffic (packets) is an important aspect of the perceived QoS. Various aspects of delay have a different impact on different services:

- End-to-end delay.
- Delay variation or Jitter.

Interactive real-time applications (e.g. voice communication) are sensitive to end-to-end delay and jitter. Long delays reduce the interactivity of the communication. Non-interactive real-time applications (e.g. one-way broadcast) are not sensitive to end-to-end delay but are affected by jitter. Jitter is usually accommodated by using a buffer at the receiver where received packets are stored and then "played back" at the appropriate time offset. The time offset (also called "playback point") is determined by maximum jitter. Applications that can adjust the playback point based on changes in the jitter value are called "adoptive" applications. Packets that arrive after their playback point has passed, are generally not useful to the application.

Non real-time applications are usually not delay-sensitive. However, because these applications may use delay measurements to control their traffic rate (e.g. TCP) or may have to buffer data until it is acknowledged (e.g. FTP), large or variable delays may affect the quality of these applications as well.

There are various components of end-to-end delay:

- Transmission delay: the time it takes to put all the bits of a packet onto the link.
- Propagation delay: the time it takes for a bit to traverse a link (usually, at the speed of light).
- Processing delay: the time it takes for a packet in a network element (e.g. routing it to the output port).
- Queuing delay: the time a packet must wait in a queue before it is scheduled for transmission.

At the endpoints, there may be additional delays in getting the packet from the network interface to the application and eventually to the user (e.g. delays in

transferring the packet across the host bus, delays in copying the packet from kernel space to user space, delays in scheduling the application).

CHAPTER IV

Meeting Delay Requirements With Wormhole Routing

In [3], Biao Chen, Hong Li, and Wei Zhao investigated the problem of wormhole routing applied to applications with real-time constraints. Their model of interconnection network was that of the Intel Paragon System. The algorithms presented in [3] assume fixed-length messages and fixed inter-arrival times. In this chapter, we write computer programs to simulate the behavior of these algorithms using variable-length messages and variable inter-arrival times.

4.1 Intel Paragon system

In the 1980s, hypercube multicomputers were made with homogeneous nodes because all I/O functions were given to the host. This limited the I/O bandwidth, and thus these computers could not be used in solving large-scale problems with efficiency or high throughput. The Intel Paragon was designed to overcome this difficulty. The usage model turns the multicomputer into an application server with multiuser access in a network environment. The Paragon system uses a simple but effective approach called X-Y routing works in the following way: a packet is first sent in the horizontal direction and then in the vertical direction. A change of direction is allowed only once in the path.

In the following, we describe the architecture of the Paragon system.

4.1.1 Node and Router Architecture

The Paragon was designed as an experimental system. Only one unit was built and delivered to Caltech in May 1991 for research use by a consortium of 13 national laboratories and universities. The Paragon is still a medium-grain multicomputer with the typical node architecture shown in figure 19.

Each router has 10 I/O ports, 5 for input and 5 for output. Four pairs of I/O channels are used for mesh connection to the four neighbors at the north, south, east, and west nodes. The fifth pair is used for internal connection between the router and local nodes. A 5x5 crossbar switch is used to establish a connection between any input channel and any output channel. Figure 20.

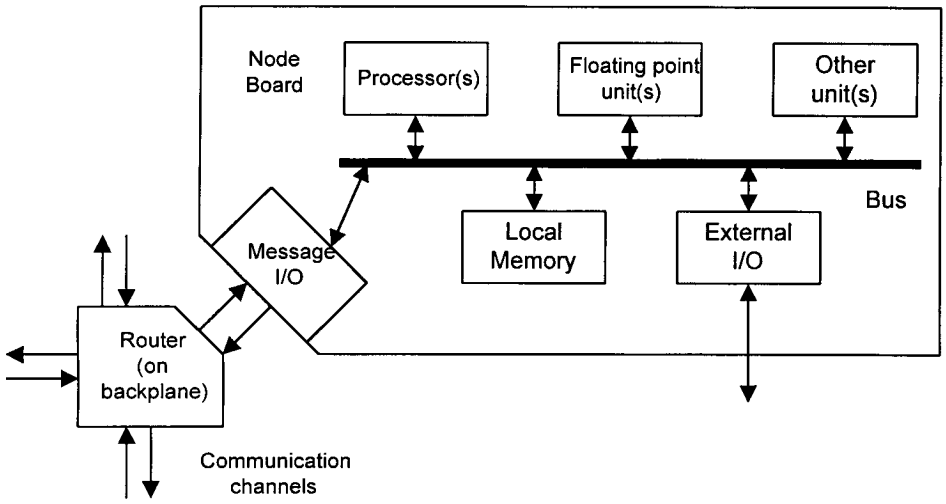


Figure 19: Intel Paragon node architecture

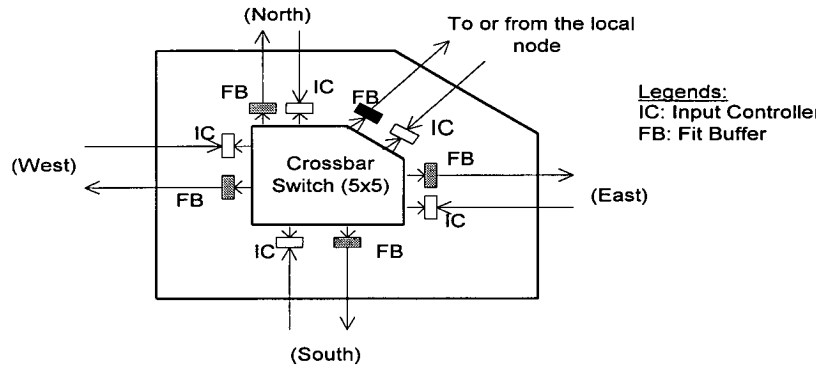


Figure 20: Intel Paragon router architecture

4.1.2 Waiting Queue

When a packet is sent from one node to another, if the channel linking these two nodes is not free, the packet has to wait for its turn. And so for other packets. Whenever packets are waiting for a channel to be free, this waiting status is called the waiting queue. And we might have a waiting queue on each node.

4.2 Message Model

In [3], the authors assumed that there are n streams of real-time messages $\{M_1, M_2, \ldots, M_n\}$ where the real-time message M_i is characterized by $M_i=(C_i, P_i, D_i, NS_i, ND_i)$ where

C_i is the length in number of flits of a message in M_i

P_i is the inter-arrival period between messages in the real-time message stream.

D_i is the maximum amount of time that may elapse between the generation time of a message at its source node and the receiving time of its last bit at the destination.

NS_i is the source node of the message stream.

ND_i is the destination node of the message stream.

In the following, i refers to the message stream i, j refers to the j^{th} message in M_i, and k refers to the k^{th} packet in the message j in M_i

The j^{th} message in stream M_i is guaranteed at time $t_{i,j} = t_{i,1} + (j-1)P_i$ where $j>=1$ and is received at the destination by $t_{i,j} + D_i$.

We also make use of the following notations:

K_i is the number of flits per packet per message.

H_i is the real packet size. $H_i = 3+K_i$. (2 header flits and 1tail flit).

N_i is the number of packets per message. It is calculated as follows:

$$N_i = \frac{C_i}{K_i}$$

W_i is the number of channels on the route of stream $M_i = ND_i - NS_i$.

$t_{i,j,k}$ is the time when the packet k becomes the first packet in the waiting queue.

$t'_{i,j,k}$ is the arrival time of the first flit of packet $M_{i,j,k}$.

$t''_{i,j,k}$ is the transmission time of the last flit of the packet $M_{i,j,k}$ from the source node.

$t'''_{i,j,k}$ is the arrival time of the last flit of packet $M_{i,j,k}$.

The end-to-end delay of the packet $M_{i,j,k}$ is the following:

$t'''_{i,j,k} - t_{i,j,k}$ = end-to-end delay of the first flit + time taken to transmit the rest of a packet.

$(t'_{i,j,k} - t_{i,j,k}) + H_i - 1$ (Hi – 1 flits follow in a pipelined fashion)

$<= W_i + H_i - 1$

$= W_i + K_i + 2$

So $W_i + H_i - 1$ is the minimum end-to-end packet delay for M_i.

Note that the transfer delay includes handshaking delay, transmission delay, propagation delay, etc, but not the blocking time. Thus, if there is no blocking on the route, it takes H units of time for a flit to pass through a route of H channels, assuming that it takes 1 unit of time to pass through 1 channel.

4.3 Transmission Control Methods

We would like to explain the two-transmission control methods, regulated and unregulated on which will be based our simulation.

The traditional transmission control method is a greedy one. Whenever the channel that connects the source node with the associated router is free, the transmission control module on the node will transmit a flit from the waiting packet into the channel. Although effective on average cases, this approach may have unfair network access in some situations.

When a packet is on the route from its source node to its destination, it can be blocked at an intermediate router because the next channel it requires has been taken by another packet. Two message steams sharing a channel on their paths may not have the same chance to access the channel even if they have the same message characteristics. A packet traveling on a long path will be in a disadvantageous position when competing for communication bandwidth with packets from a short path. This is simply because the header flit of the long path packet has to go through more routers and has to wait for its turn on all these routers before the whole packet can go through. This will result in rate disparity and unfairness of network access.

To further illustrate this point, let us consider an example. Four nodes are involved in this example. We assume an extreme case that each node generates messages sufficiently fast so that whenever the associated channel is available, a packet is ready to be transmitted.

N1, N2 and N3 each send a flow of packets into the network. We label the packets from N1 as A1, A2, ... Similarly, we label the packets from N2 and N3 as B1, B2, ..., and C1, C2, ..., respectively. The destination node of all the packets is N4.

Packets from node N1 may be blocked by those from node N2 at router R2. Packets from both nodes N1 and N2 may be blocked by those from node N3 at router R3. This what will happen: A1 is blocked by B1 at R2 while B1 is blocked by C1 and R3. This is because C1 is holding channel L3 and B1 is holding channel L2. At the

time where the last flit of C1 arrives at N4, B1 obtains access to L3. While flits of B1 are being transmitted across channel L3, the first header flit of C2 arrives at R3. As the last flit of B1 leaves R2, L2 becomes available to A1. Now both C2 and A1 are waiting for L3. According to Round Robin scheduling algorithm, L3 will be granted to C2. After C2 releases L3, A1 can eventually get L4 and reach N4. Therefore we should observe that on L3, the packets appear in the following order: C1, B1, C2, A1, C3, B2, C4, A2... Thus, the packets from N3 appear on channel L3 twice as often as those from N1 or N2. (Figure 21).

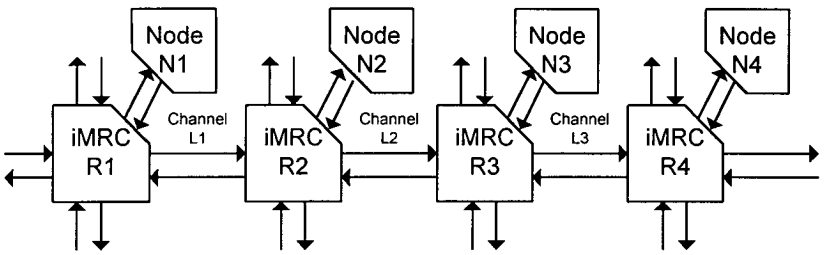


Figure 21: Traditional transmission control method

This will cause the unfairness of network access, and a poor performance in terms of meeting message deadlines.

To overcome this problem, the authors in [3] suggested the regulated method where they allow holding some packets for a while before transmitting them. This will be done by using a token to manage the transmissions. Flits of a packet waiting at the waiting queue can be transmitted if and only if:

- The channel that connects to the associated router is free and
- There is a token.

The token is generated in the following manner:

- Initially there is a token at the transmission control module.
- If a token is discarded at time t, the next token will be generated at time t + TP.

This will reduce the unfairness of network access, resulting in a better network performance in terms of meeting message deadlines.

The objective of this paper is to analyze message transmission control methods that aim meeting message's delay requirements.

Now, for both cases, the Regulated and Unregulated method, messages should arrive to destination before their deadline D_i.

So for the Regulated method, as discussed earlier, we should have a token to schedule the time of packets. So the algorithm of the regulated will be as follows:

First, we let TP (Token Generation Interval) be the minimum of all message deadlines so at least one packet from a message can be transmitted before the deadline. So TP is the time elapsed between the transmission of one packet and another from the same message.

Second, in order for the messages to reach their destination before their deadline, we should bound the packet delay within the token generation interval (TP). In this case, whenever a new token is generated, a new packet can start its transmission. So within a time interval of Di, [Di/TP] packets can be transmitted.

So the number of flits in a packet will be calculated as follows:

$$K_i = \frac{C_i}{[\frac{D_i}{TP}]}$$

As for the Unregulated method, no token is needed. The first flit of the first packet of a certain queue will be transmitted whenever the channel is free. Note that in this method, the blocking time at node i will be equal to the blocking time at node i+1 and the holding time at node i.

4.4 Algorithms A1 and A2 Applied to Variable C_i and P_i.

A. Algorithm A1

The following is the pseudocode for the regulated method. In [3], the authors have done their experiments on a 2D-Mesh architecture. In our simulation we use the same algorithm except that C_i and P_i are variables and a simple linear array architecture is used.

Input: Set $M=\{M_1, M_2, \ldots, M_n\}$
Output: TP and K_i, i=1,2,…,n.
Begin
/*Select token generation period*/
/*Choose packet size K_i for each stream*/

$$TP = \min_{i=1}^{n}(D_i)$$

$$K_i = [\frac{C_i}{[\frac{D_i}{TP}]}]$$

for i=1 to n do
Endfor
End.

First we need to generate messages. When a message is generated, files are created in which we have the length of the message as C_i (chosen randomly), the inter-arrival period P_i (chosen randomly), the deadline of the every message D_i (randomly chosen), its source node NS_i and its destination node ND_i (randomly chosen).

While running the program, the messages from a file are read. Once read, the messages are divided into packets which is done as follows:

Every message C_i is divided into N_i packets, each of K_i flits determined as follows:

$$K_i = [\frac{C_i}{[\frac{D_i}{TP}]}]$$

where TP is the token generation period computed as

$$TP = \min_{i=1}^{n}(D_i)$$

Thus $N_i = Ci/K_i$.

Every P_i a message M_i is generated and packets are sent to their corresponding source node along with the packet number, the source address, the destination address, the message deadline, the inter-arrival period, and the message sent-time.

If the last packet of M_i is at ND_i and a TP is generated than the arrival time is set. The time elapsed between the generation of the message and the arrival time of its last packet to destination will show whether the message has reached its destination within its deadline D_i or not.

If, at any node I, a packet k exists and TP is generated, this packet is sent to either node i+1 or node i-1 depending on its destination until it reaches NDi where it will be removed totally.

At the end of the simulation, we would be able to know how many messages arrived to their destination within their deadlines and how many messages missed their deadlines.

In figure 22 below, we show a snapshot of A1 in execution.

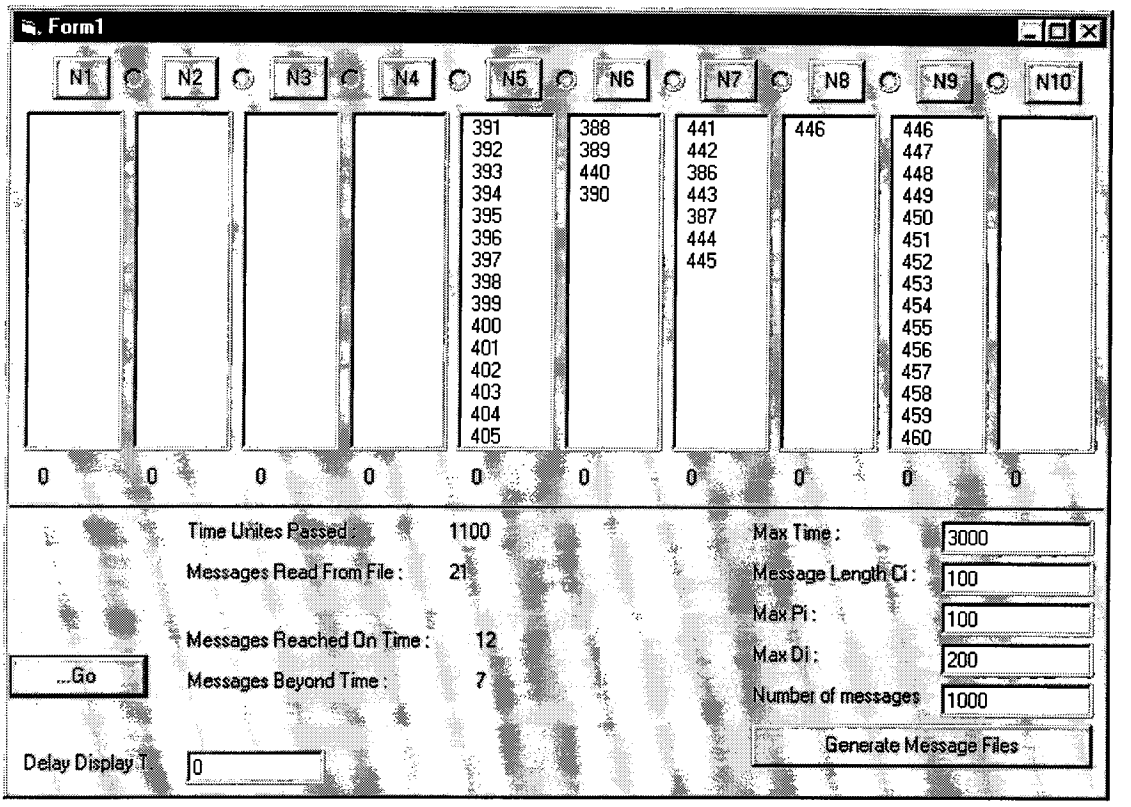


Figure 22: A1 running algorithm

B. Algorithm A2

The following is the pseudocode for the unregulated method. In [3], the authors have done their experiments on a 2D-Mesh architecture. In our simulation we will use the same algorithm except that C_i and P_i will be variables and on a simple linear array architecture.

Input: Set $M=\{M_1, M_2, \ldots, M_n\}$
Output: K_i, i=1,2,…,n.
Begin
    For i = 1 to n do
        $K_i = 1$;
    Endfor
    Repeat
        Check the following inequality
        Let i be the first index that the corresponding inequality is not satisfied.

$$N_i * (\sum_{l \in R_i} B_{(l, M_i)}(\vec{K}) + W_i + H_i - 1) \leq D_i$$

Begin

$K_i = K_i + 1$;

If ($K_i > C_i$) then

Return with no solution;

Endif

End

Until (no K_i is modified);

Return all K_i as the solution;

End

The procedure concerning generation of files is similar to the previous algorithm. But when it comes to dividing the message into packets, we do not need TP in order to get K_i. Here we use the inequality used in [3].

$$N_i * (\sum_{l \in R_i} B_{(l, M_i)}(\vec{K}) + W_i + H_i - 1) \leq D_i$$

where B = max b and b is the blocking time of every packet at each node.

The inequality should be satisfied in order to know what is K_i. First we initialize K_i to 1 for all messages. So initially $N_i = C_i/K_i = C_i$. $W_i = (ND_i - NS_i)$. $H_i = K_i + 3$. Given D_I, the only thing that we need to calculate is B. The blocking time at node i is equal to the blocking time at node i + 1 added to it the holding time at node i. The blocking time at the destination node is 0. We assume that the holding time at each node is $K_i + 2$.

Consider for example a packet with source node 2 and destination node 5, the blocking time at each node is calculated as follows:

At node 5, B= 0, at node 4, $B = B_5 + h_4 = 0+K_i+2 = K_i+2$, at node 3,

$B = B_5 + B_4 + h_3 = 0 + K_i+2 + K_i+2 = 2(K_i+2)$,

at node 2, $B = B_5 + B_4 + B_3 + h_2 = 0 + K_i+2 + K_i+2 + K_i+2 = 3(K_i+2)$.

$B = (ND_i - NS_i)x(K_i+2)$.

If the inequality is satisfied, we find K_i and we divide the message into packets using $N_i = C_i/K_i$.

If the inequality is not satisfied, we increment K_i by 1 until it is satisfied or until K_i becomes greater than C_i. If K_i is greater than C_i than the message is not sent.

Once the messages are divided into packets, these packets are sent to their source nodes. At destination node, if the packet existing is the last packet of Mi, than the arrival time is set. The time elapsed between the generation of the message and the arrival time of its last packet will indicate whether the message reached its destination within its deadline Di or not.

If at any node i, a packet k exists, its blocking time will be calculated as follows: $(ND_i$ – Current node)x(K_i+2) . At every unit of time, the blocking time at node i is decremented by 1 until its blocking time gets to 0 than the packet can be sent to either node i+1 or node i-1 depending on its destination. This is repeated until the packet reaches its destination where it will be removed totally.

At the end of the simulation, we would be able to know how many messages arrived to their destination within their deadlines, how many messages missed their deadlines and how many messages were unable to be sent for K_i is greater than C_i.

In figure 23, we show a snapshot of A2 running where K_i is greater than C_i.

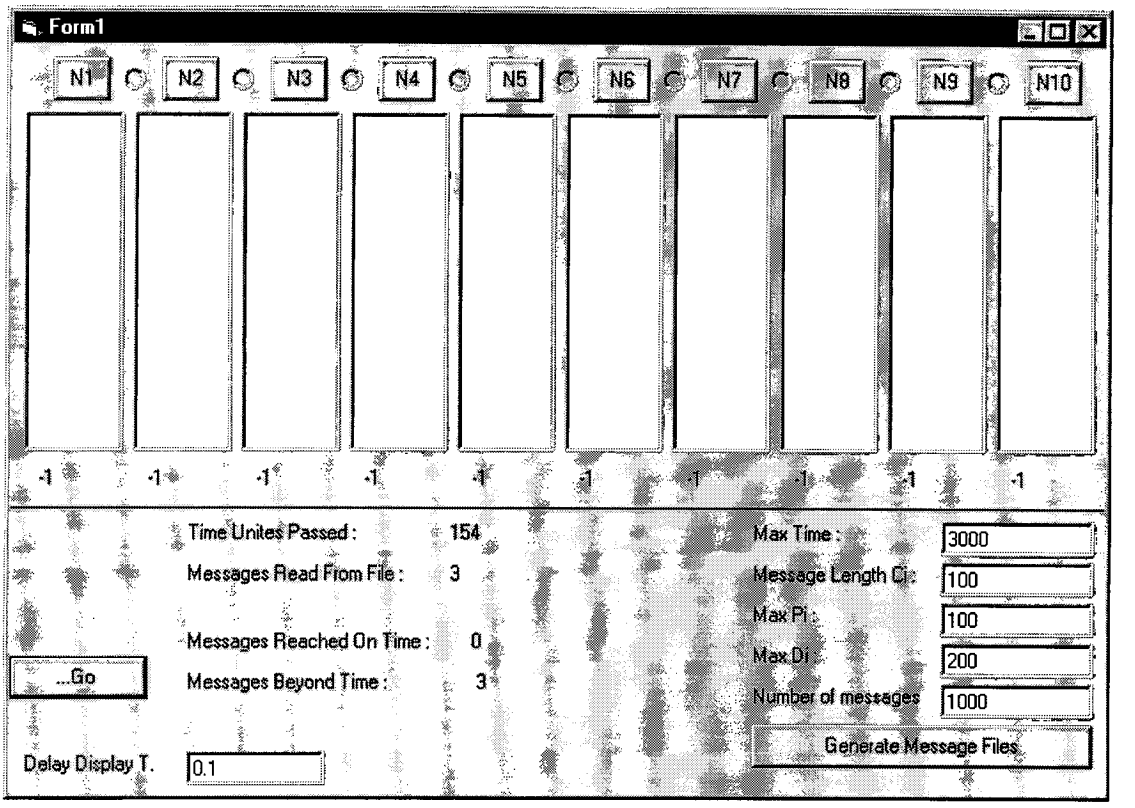


Figure 23: A2 running algorithm with $K_i > C_i$

In figure 24, we show a snapshot of A2 in execution.

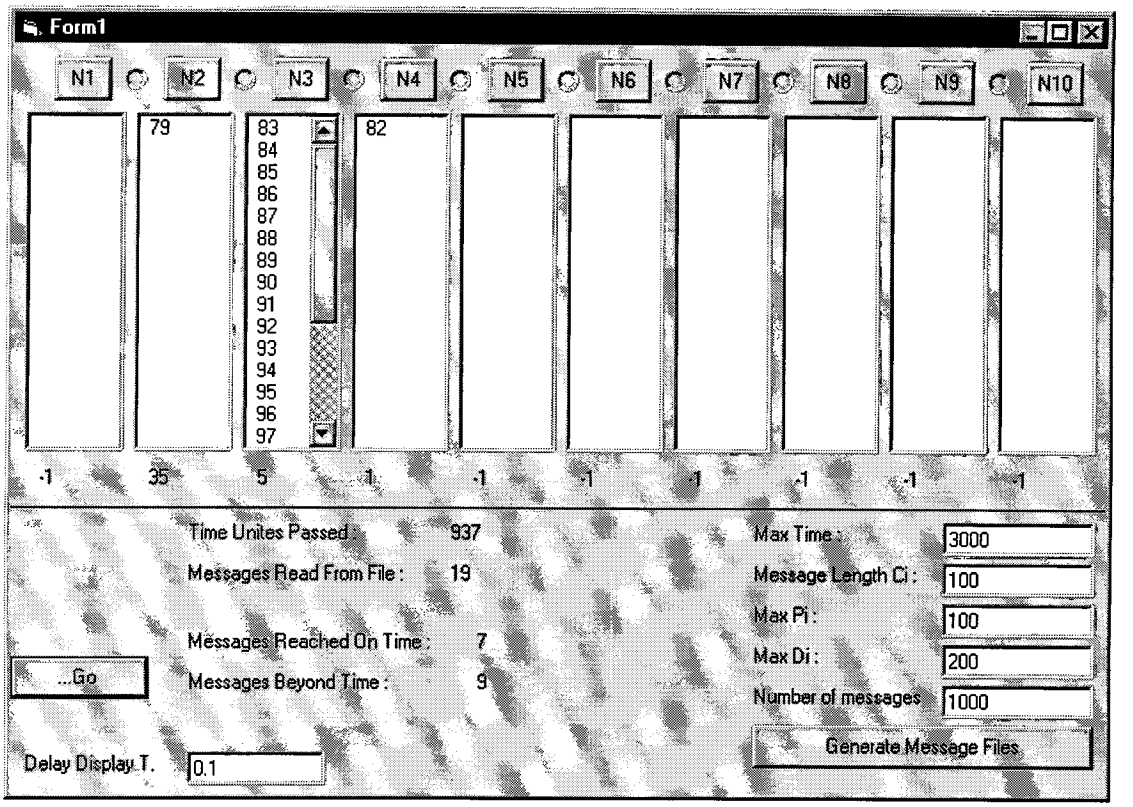


Figure 24: A2 running algorithm

The actual code of both algorithms can be found in Appendix A.

The results of the simulation are presented in chapter V.

CHAPTER V

Simulation Results

This chapter summarizes the results of our simulation.

Algorithm A1

We set the running time to be 3000 units of time. We assume that D_i is 200 units of time and a maximum number of messages of 1000.

Table 2 below shows the results of our simulation for different values of C_i and P_i.

Pi \ Ci	25	50	100	150	200
100	0.9	0.81	0.55	0.29	0.19
150	0.89	0.87	0.55	0.45	0.18
200	0.86	0.84	0.6	0.42	0.21

Table 2: A1 Routing efficiency function of C_i & P_i

Pictorially put, the results are as shown below:

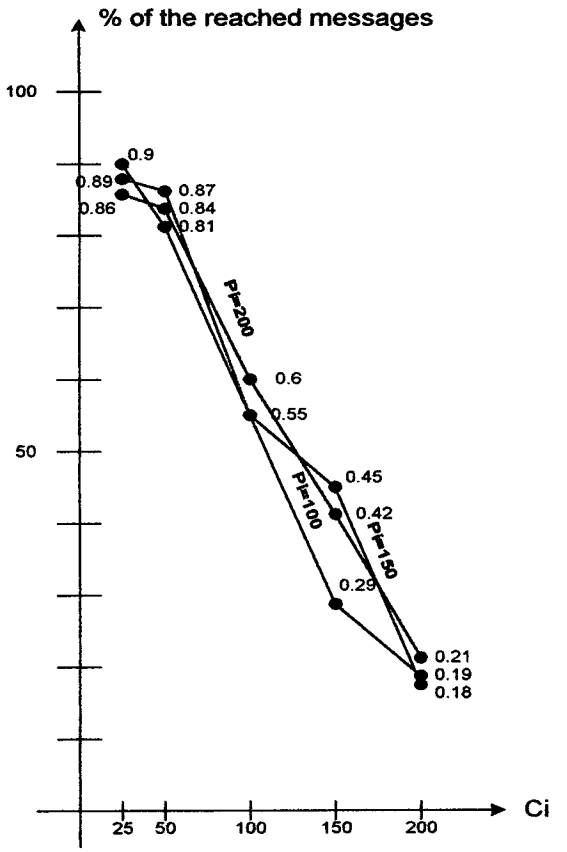


Figure 24: A1 Routing efficiency function of C_i & P_i

Algorithm A2

We set the running time to be 3000 units of time. We assume that D_i is 200 units of time and a maximum number of messages of 1000.

Table 3 below shows the results of our simulation for different values of C_i and P_i.

Pi \ Ci	25	50	100	150	200
100	0.57	0.29	0.38	0.24	0.19
150	0.65	0.41	0.41	0.35	0.2
200	0.67	0.48	0.34	0.35	0.2

Table 3: A2 Routing efficiency function of C_i & P_i

Pictorially put, the results are as shown below:

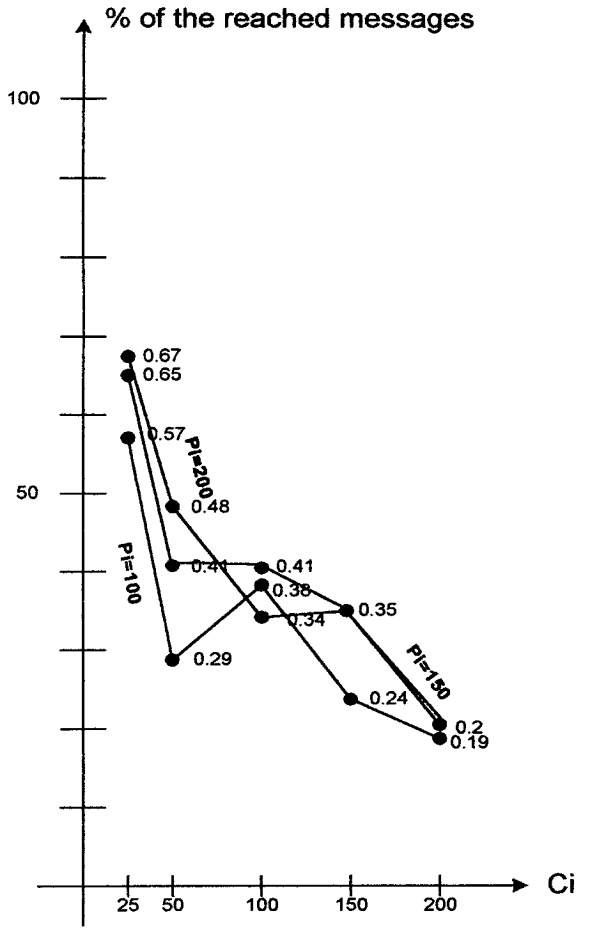


Figure 25: A2 Routing efficiency function of C_i & P_i

It is clear from figures 25 and 26 that the regulated transmission control method gives a better result than the unregulated transmission control method which generalizes the algorithms in [3] to networks with variable length messages and inter-arrival periods.

CONCLUSION AND FUTURE RESEARCH

In this thesis, we have discussed the issue of meeting delay requirements in computer networks using wormhole routing. In [3], the authors showed that if they regulate the rate of transmission at each source node, over 50% of messages would reach their deadlines. They used message streams of fixed length messages and fixed inter-arrival times using a 2D-Mesh architecture. In this thesis, we investigated the performance of the algorithms presented in [3] in a computer network with message streams of variable length messages and variable inter-arrival times using a simple linear array architecture. Our simulation showed that the regulated transmission control continues to provide better results than the unregulated transmission method.

Many extensions of this study are possible. The idea of controlling transmission at the source node is definitely worth further investigation. Additional research in this direction may consider the performance of the same algorithms applied to different architectures with different parameters or the creation of an entirely different technique than the token generation method adopted in this thesis.

APPENDIX A

A1 Algorithm

This program implements the regulated transmission control method applied to variable length messages and inter-arrival periods.

```
'Written in VB version 5.0
Option Explicit
Const MaxNode = 10
'Const MaxTime = 20000  'Units of time
Const NMessages = 4000
Dim Channel(MaxNode - 1) As Integer
Dim NumberOfMessages As Long

Dim MaxTime As Long


'Declare the message info
Dim Msg1 As String ' contains the message
Dim Ci As Integer ' Length of Message # of Flits
Dim Pi As Integer ' Inter-arrival Period
Dim Di As Integer ' Deadline for the message reach destination
Dim Sn As Integer ' Sourse Node
Dim Dn As Integer ' Destination Node
Dim TP As Integer ' Token Generation Period
Dim MessageSendTime As Long

'This will help us track the packets to know which packet ends a message
Dim PacketNumber As Long

'The nodecol corresponds to the list of packets waiting
' on the corresponding nodes.
Dim NodeCol(1 To MaxNode) As New Collection

'The Token generation period for the packets on every node
Dim NodeTP(1 To MaxNode) As Integer

'Holding the message information
Dim MessageInfo(1 To NMessages) As New Collection


'The counter of my time
Dim TimeCount As Long

'Message Pos is for knowing the message number where the last packet belongs to inside the message
info
Dim MessagePos As Long



Sub main()

   'Read the data from the file
   Dim strfilter As String
   Dim strFileName As String
```

```
Dim DefaultName As String
Dim FileStrLen As Integer

'The file management related material
'strfilter = "Textstr (*.str)|*.str|All Files (*.*)|*.*"
strfilter = "All Files (*.*)|*.*"
'strfilter = "TextStr (*.str)|*.str"

CDlg1.Filter = strfilter
'Open the common dialog in open mode
CDlg1.DialogTitle = "Open the Str file ... "
CDlg1.filename = "c:\tony\Rania B\proj under VB5\*.*"
CDlg1.ShowOpen
'Make sure the retrieved filename is not a blank string
If CDlg1.filename = "" Then
   MsgBox "Invalid File Name !"
   Exit Sub
End If
'If it is not blank, open the file

'Get the name without the path
'strFileName = CDlg1.FileTitle
'Dim FileExt As String
'FileExt = Right(strFileName, 4)
'MsgBox FileExt
'If FileExt <> ".str" Then
'   MsgBox "Invalid file name !!!"
'   Exit Sub
'End If

If Not (CDlg1.filename Like "*.str") Then
   MsgBox "Invalid file name, *.str expected... !!!"
   Exit Sub
End If


'Get the original file name without the extension
FileStrLen = Len(CDlg1.filename)
DefaultName = Left(CDlg1.filename, FileStrLen - 4)
'MsgBox DefaultName

'Get the token generation period for the given file input
GetTP (DefaultName)


'Open the files in request
Open (DefaultName + ".str") For Input As #1
Open (DefaultName + ".pi") For Input As #2
Open (DefaultName + ".di") For Input As #3
Open (DefaultName + ".sn") For Input As #4
Open (DefaultName + ".dn") For Input As #5


'Initialize the packet number for tracing packets
PacketNumber = 0
MaxTime = Val(Text5.Text)
Dim i, j As Integer
For i = 1 To MaxNode
   TPVal(i).Caption = "0"
   NodeTP(i) = 0
```

```
    'free the nodes from any packets
    For j = 1 To NodeCol(i).Count
       NodeCol(i).Remove 1
    Next j

    For j = 1 To List1(i).ListCount
       List1(i).RemoveItem 0
    Next j

Next i

Dim tmpval As Integer
'free the message info if any
tmpval = Val(Label3.Caption)
For i = 1 To tmpval
   For j = 1 To MessageInfo(i).Count
      MessageInfo(i).Remove 1
   Next j
Next i
Label3.Caption = "0"
Label4.Caption = "0"
Label8.Caption = "0"
Label9.Caption = "0"



GetNewMessageInfo
DivideMessageIntoPackets
'MessageSendTime = TimeCount 'Get the time of sending the message

For TimeCount = 1 To MaxTime
   Label4.Caption = Str(TimeCount)
   Label4.Refresh
   CheckPackets    'checks the packets
   'MsgBox Pi, , "PI"

   If (Pi = 0) Then
      If (Not EOF(1)) Then
         GetNewMessageInfo
         DivideMessageIntoPackets
      End If
      'MessageSendTime = TimeCount 'Get the time of sending the message
      ' the get message info will get the new Pi as well
   Else
      Pi = Pi - 1
   End If

   '/*************
   Dim PauseTime As Double
   Dim start As Double
   PauseTime = Text6.Text
   start = Timer   ' Set start time.
   Do While Timer < start + PauseTime
      'do nothing
   Loop
   '/*************

   Next TimeCount

'Close the file when completed
```

```
    Close #1
    Close #2
    Close #3
    Close #4
    Close #5

End Sub

Sub GetNewMessageInfo()
    Dim xStr As String
    Dim xInt As Integer

    Line Input #1, Msg1
    'MsgBox Msg1, , "Message"
    Ci = Len(Msg1)
    'MsgBox Ci, , "Ci"

    Line Input #2, xStr
    Pi = Val(xStr)
    'MsgBox Pi, , "Pi"

    Line Input #3, xStr
    Di = Val(xStr)
    'MsgBox Di, , "Di"

    Line Input #4, xStr
    Sn = Val(xStr)
    'MsgBox Sn, , "Sn"

    Line Input #5, xStr
    Dn = Val(xStr)
    'MsgBox Dn, , "Dn"

    'Setting the display representing the number of messages read
    Label3.Caption = Str(Val(Label3.Caption) + 1)
    Label3.Refresh
End Sub

Private Sub Command6_Click()
    main
End Sub

Sub DivideMessageIntoPackets()

    ' store the send time of the message

    Dim Ki As Integer
    Dim NoOfPackets As Integer
    Dim PacketString As String
    Dim i As Integer
    PacketString = ""

    'Get the packet size Ki
    Ki = Ci / (Di / TP)
    'MsgBox Ki, , "Ki"
    If Ki = 0 Then
        NoOfPackets = 1
    Else
        'Get the number of packets in this message
        NoOfPackets = Ci / Ki
```

```
    'MsgBox NoOfPackets, , "No of Packets"
  End If

  'Fill the packet string
  Dim LocalStr As String
  LocalStr = Msg1
  For i = 1 To NoOfPackets - 1
    'First fill the packet Number
    PacketNumber = PacketNumber + 1
    PacketString = Str(PacketNumber) + "***"
    'Fill the destination node
    PacketString = PacketString + Str(Dn) + "***"
    'Fill the data ki characters
    PacketString = PacketString + Left(LocalStr, Ki)
    LocalStr = Right(LocalStr, Len(LocalStr) - Ki)

    'Fill the packet in the correspondign node waiting list
    NodeCol(Sn).Add PacketString
    List1.Item(Sn).AddItem Str(PacketNumber)
  Next i

  'Fill the last packet in this string
  'First fill the packet Number
  PacketNumber = PacketNumber + 1
  PacketString = Str(PacketNumber) + "***"
  'Fill the destination node
  PacketString = PacketString + Str(Dn) + "***"
  'Fill the data ki characters
  PacketString = PacketString + LocalStr
  'Set the wanted info in the message such as packet no ...

  MessageInfo(Val(Label3.Caption)).Add TimeCount
  MessageInfo(Val(Label3.Caption)).Add Sn
  MessageInfo(Val(Label3.Caption)).Add Dn
  MessageInfo(Val(Label3.Caption)).Add PacketNumber
  MessageInfo(Val(Label3.Caption)).Add Di
  MessageInfo(Val(Label3.Caption)).Add Pi

  'Fill the packet in the correspondign node waiting list
  NodeCol(Sn).Add PacketString
  List1.Item(Sn).AddItem Str(PacketNumber)

End Sub

Private Sub Command7_Click()

  Dim i As Integer
  Dim myString As String
  Dim CharVal As Integer
  Dim strfilter As String
  Dim strFileName As String
  Dim msgCount As Integer
  Dim RetVal As Variant

  'The file management related material
  'strfilter = "StrText (*.str)|*.str|All Files (*.*)|*.*"
  strfilter = "All Files (*.*)|*.*"
```

```
CDlg1.Filter = strfilter
'Open the common dialog in save mode
CDlg1.filename = ""
CDlg1.ShowSave
'Make sure the retrieved filename is not a blank string
If CDlg1.filename = "" Then
   MsgBox "Invalid File Name !"
   Exit Sub
End If

'MsgBox (CDlg1)
'Exit Sub

'If it is not blank, open the file
strFileName = CDlg1.filename


'Open the files for entry
'Open a file for writing strings in it
Open (strFileName + ".str") For Output As #1
'Open a file for writing Pi in it
Open (strFileName + ".pi") For Output As #2
'Open a file for writing Di in it
Open (strFileName + ".di") For Output As #3
'Open a file for writing Sn in it
Open (strFileName + ".sn") For Output As #4
'Open a file for writing Dn in it
Open (strFileName + ".dn") For Output As #5


'Fill the message data in the respective files
Form1.MousePointer = vbHourglass
TP = 10 'max of di value

NumberOfMessages = Val(Text1.Text)
For msgCount = 1 To NumberOfMessages
   '*****
   'make the string and put it in the file
   Ci = Rnd * Val(Text4.Text)
   If Ci = 0 Then Ci = 1
   'MsgBox Ci, , "Ci"
   myString = ""
   For i = 1 To Ci
      CharVal = (Rnd * 100) + 30 'The +30 is just to skip the special characters
      myString = myString + Chr(CharVal)
   Next i
   'MsgBox myString, , "MyStirng Value is ..."
   'Do the write
   Print #1, myString
   '*****

   'Time between message and another
   Pi = (Rnd * Val(Text3.Text)) 'The + 1 is for the first time it is called to replace the -1
   If Pi = 0 Then Pi = 1
   'MsgBox Pi, , "PI"
   'Do the write
   Print #2, Pi
   '*****

   'Deadline for the message
```

```
        Di = Rnd * Val(Text2.Text)
        If Di = 0 Then Di = 1
        'MsgBox Di, , "Di"
        'Do the write
        Print #3, Di

        If Di < TP Then
           TP = Di
        End If
        '*****

        Sn = Rnd * MaxNode  'the limit is the MaxNode
        If Sn = 0 Then Sn = 1
        'MsgBox Sn, , "Sn"
        'Do the write
        Print #4, Sn
        '*****

        Dn = Rnd * MaxNode  'the limit is the MaxNode
        If Dn = 0 Then Dn = 1
        'MsgBox Dn, , "Dn"
        'Do the write
        Print #5, Dn
        '*****
    Next msgCount
    Form1.MousePointer = vbDefault


    MsgBox TP, , "TP"
    'Close the file when completed
    Close #1
    Close #2
    Close #3
    Close #4
    Close #5

End Sub


Sub CheckPackets()
    Dim i As Integer
    Dim LastPacket As Boolean
    Dim PacketNo As Long
    Dim packetdn As Integer
    Dim CurrPacket As String
    Dim SubPacket As String
    Dim StarPos As Integer

    For i = 1 To MaxNode

        If NodeTP(i) = 0 Then
           'GetPacketInfo done here
           If NodeCol(i).Count <> 0 Then

              CurrPacket = NodeCol(i).Item(1)
                 'MsgBox CurrPacket, , "Current Packet Value"
              StarPos = InStr(CurrPacket, "***")
                 'MsgBox StarPos - 1
              PacketNo = Val(Left(CurrPacket, StarPos - 1))
                 'MsgBox PacketNo, , "packet No in packet"
```

```
        SubPacket = Right(CurrPacket, Len(CurrPacket) - (StarPos - 1) - 3)
          'MsgBox SubPacket, , "Sub packet"
        StarPos = InStr(SubPacket, "***")
          'MsgBox StarPos - 1
        packetdn = Val(Left(SubPacket, StarPos - 1))
          'MsgBox packetdn, , "packet destination in packet"



        If (i = packetdn) Then

          CheckLastPacket (PacketNo)
          If MessagePos <> -1 Then
            LastPacket = True
          Else
            LastPacket = False
          End If

          If (LastPacket) Then
            'Set the arrival time in the message info
            'MsgBox (TimeCount - MessageInfo(MessagePos).Item(1))
            'MsgBox MessageInfo(MessagePos).Item(5)
            If ((TimeCount - MessageInfo(MessagePos).Item(1)) <
MessageInfo(MessagePos).Item(5)) Then
              'messages arrived before deadline
              Label8.Caption = Str(Val(Label8.Caption) + 1)
              Label8.Refresh
            Else
              'messages arrived after message deadline
              Label9.Caption = Str(Val(Label9.Caption) + 1)
              Label9.Refresh
            End If


            '***********
            NodeCol(i).Remove 1
            List1(i).RemoveItem (0)
            List1(i).Refresh
          Else
            NodeCol(i).Remove 1
            List1(i).RemoveItem (0)
            List1(i).Refresh
          End If
          NodeTP(i) = TP
          TPVal(i).Caption = NodeTP(i)
          TPVal(i).Refresh
        Else
          If (i < packetdn) Then
            NodeCol(i + 1).Add (NodeCol(i).Item(1))
            List1(i + 1).AddItem List1(i).List(0)
            List1(i + 1).Refresh
            NodeCol(i).Remove 1
            List1(i).RemoveItem (0)
            List1(i).Refresh
            NodeTP(i) = TP
            TPVal(i).Caption = NodeTP(i)
            TPVal(i).Refresh
            If List1(i + 1).ListCount = 1 Then
              i = i + 1
            Else
```

```
            End If
          Else
            NodeCol(i - 1).Add (NodeCol(i).Item(1))
            List1(i - 1).AddItem List1(i).List(0)
            List1(i - 1).Refresh
            NodeCol(i).Remove 1
            List1(i).RemoveItem (0)
            List1(i).Refresh
            NodeTP(i) = TP
            TPVal(i).Caption = NodeTP(i)
            TPVal(i).Refresh
          End If
        End If
      End If
    Else
      NodeTP(i) = NodeTP(i) - 1
      TPVal(i).Caption = NodeTP(i)
      TPVal(i).Refresh

    End If
  Next i
End Sub


Sub GetTP(DIfname As String)
  Dim xStr As String


  TP = 10 'max possible value for tp
  Open (DIfname + ".di") For Input As #9
    While Not EOF(9)

      Line Input #9, xStr
      Di = Val(xStr)
      'MsgBox Di, , "Di"

      If Di < TP Then
        TP = Di
      End If

    Wend
  'Close the file when completed
  Close #9

  'MsgBox TP
End Sub

Sub CheckLastPacket(pacNo As Long)
  Dim i As Integer

  For i = 1 To Val(Label3.Caption)
    If (MessageInfo(i).Item(4) = pacNo) Then
      MessagePos = i
      Exit Sub
    End If
  Next i
  MessagePos = -1
End Sub
```

APPENDIX B

A2 ALGORITHM

This program implements the unregulated transmission control method applied to variable length messages and inter-arrival periods.

```
'Written in VB version 5.0
Option Explicit
Const MaxNode = 10
'Const MaxTime = 20000  'Units of time
Const NMessages = 4000
Dim Channel(MaxNode - 1) As Integer
Dim NumberOfMessages As Long

Dim MaxTime As Long


'Declare the message info
Dim Msg1 As String ' contains the message
Dim Ci As Integer ' Length of Message # of Flits
Dim Pi As Integer ' Inter-arrival Period
Dim Di As Integer ' Deadline for the message reach destination
Dim Sn As Integer ' Sourse Node
Dim Dn As Integer ' Destination Node
'Dim TP As Integer ' Token Generation Period
Dim MessageSendTime As Long

'This will help us track the packets to know which packet ends a message
Dim PacketNumber As Long

'The nodecol corresponds to the list of packets waiting
' on the corresponding nodes.
Dim NodeCol(1 To MaxNode) As New Collection

'not used in unRegulated
'The Token generation period for the packets on every node
Dim NodeB(1 To MaxNode) As Integer

'Holding the message information
Dim MessageInfo(1 To NMessages) As New Collection


'The counter of my time
Dim TimeCount As Long

'Message Pos is for knowing the message number where the last packet belongs to inside the message
info
Dim MessagePos As Long



Sub Main()

  'Read the data from the file
  Dim strfilter As String
```

```
Dim strFileName As String
Dim DefaultName As String
Dim FileStrLen As Integer

'The file management related material
'strfilter = "Textstr (*.str)|*.str|All Files (*.*)|*.*"
strfilter = "All Files (*.*)|*.*"
'strfilter = "TextStr (*.str)|*.str"

CDlg1.Filter = strfilter
'Open the common dialog in open mode
CDlg1.DialogTitle = "Open the Str file ... "
CDlg1.filename = "c:\tony\Rania B\proj under VB5\*.*"
CDlg1.ShowOpen
'Make sure the retrieved filename is not a blank string
If CDlg1.filename = "" Then
   MsgBox "Invalid File Name !"
   Exit Sub
End If
'If it is not blank, open the file

'Get the name without the path
'strFileName = CDlg1.FileTitle
'Dim FileExt As String
'FileExt = Right(strFileName, 4)
'MsgBox FileExt
'If FileExt <> ".str" Then
'   MsgBox "Invalid file name !!!"
'   Exit Sub
'End If

If Not (CDlg1.filename Like "*.str") Then
   MsgBox "Invalid file name, *.str expected... !!!"
   Exit Sub
End If


'Get the original file name without the extension
FileStrLen = Len(CDlg1.filename)
DefaultName = Left(CDlg1.filename, FileStrLen - 4)
'MsgBox DefaultName

'Open the files in request
Open (DefaultName + ".str") For Input As #1
Open (DefaultName + ".pi") For Input As #2
Open (DefaultName + ".di") For Input As #3
Open (DefaultName + ".sn") For Input As #4
Open (DefaultName + ".dn") For Input As #5


'Initialize the packet number for tracing packets
PacketNumber = 0
MaxTime = Val(Text5.Text)
Dim i, j As Integer
For i = 1 To MaxNode
   BVal(i).Caption = "-1"
   NodeB(i) = -1
   'free the nodes from any packets
   For j = 1 To NodeCol(i).Count
      NodeCol(i).Remove 1
```

```
    Next j

    For j = 1 To List1(i).ListCount
       List1(i).RemoveItem 0
    Next j
 Next i

 Dim tmpval As Integer
 'free the message info if any
 tmpval = Val(Label3.Caption)
 For i = 1 To tmpval
    For j = 1 To MessageInfo(i).Count
       MessageInfo(i).Remove 1
    Next j
 Next i
 Label3.Caption = "0"
 Label4.Caption = "0"
 Label8.Caption = "0"
 Label9.Caption = "0"



 GetNewMessageInfo
 DivideMessageIntoPackets
 'MessageSendTime = TimeCount 'Get the time of sending the message

 For TimeCount = 1 To MaxTime
    Label4.Caption = Str(TimeCount)
    Label4.Refresh
    CheckPackets    'checks the packets
    'MsgBox Pi, , "PI"

    If (Pi = 0) Then
       If (Not EOF(1)) Then
          GetNewMessageInfo
          DivideMessageIntoPackets
       End If
       'MessageSendTime = TimeCount 'Get the time of sending the message
       ' the get message info will get the new Pi as well
    Else
       Pi = Pi - 1
    End If

    '/*************
    Dim PauseTime As Double
    Dim start As Double
    PauseTime = Text6.Text
    start = Timer   ' Set start time.
    Do While Timer < start + PauseTime
       'do nothing
    Loop
    '/*************

    Next TimeCount

 'Close the file when completed
 Close #1
 Close #2
 Close #3
 Close #4
```

```
    Close #5

End Sub

Sub GetNewMessageInfo()
    Dim xStr As String
    Dim xInt As Integer

    Line Input #1, Msg1
    'MsgBox Msg1, , "Message"
    Ci = Len(Msg1)
    'MsgBox Ci, , "Ci"

    Line Input #2, xStr
    Pi = Val(xStr)
    'MsgBox Pi, , "Pi"

    Line Input #3, xStr
    Di = Val(xStr)
    'MsgBox Di, , "Di"

    Line Input #4, xStr
    Sn = Val(xStr)
    'MsgBox Sn, , "Sn"

    Line Input #5, xStr
    Dn = Val(xStr)
    'MsgBox Dn, , "Dn"

    'Setting the display representing the number of messages read
    Label3.Caption = Str(Val(Label3.Caption) + 1)
    Label3.Refresh
End Sub

Private Sub Command6_Click()
    Main
End Sub

Sub DivideMessageIntoPackets()

    ' store the send time of the message

    Dim Ki As Integer
    Dim NoOfPackets As Integer
    Dim PacketString As String
    Dim i As Integer
    PacketString = ""

    'Get the packet size Ki
    'Ki = Ci / (Di / TP)

    'Obtaining the ki in the unregulated case.
    Ki = GetKi


    'MsgBox Ki, , "Ki"
    If Ki = -1 Then
        ' the message did not reach the destination
        ' increase the number of undelivered messages by one
```

```
    Label9.Caption = Str(Val(Label9.Caption) + 1)
    Label9.Refresh

    'add message info
    MessageInfo(Val(Label3.Caption)).Add TimeCount
    MessageInfo(Val(Label3.Caption)).Add Sn
    MessageInfo(Val(Label3.Caption)).Add Dn
    MessageInfo(Val(Label3.Caption)).Add PacketNumber
    MessageInfo(Val(Label3.Caption)).Add Di
    MessageInfo(Val(Label3.Caption)).Add Pi

    'exit the funciton without sending the packets
    Exit Sub
Else
    'Get the number of packets in this message
    NoOfPackets = Ci / Ki
    'MsgBox NoOfPackets, , "No of Packets"
End If

'Fill the packet string
Dim LocalStr As String
LocalStr = Msg1
For i = 1 To NoOfPackets - 1
    'First fill the packet Number
    PacketNumber = PacketNumber + 1
    PacketString = Str(PacketNumber) + "***"
    'Fill the destination node
    PacketString = PacketString + Str(Dn) + "***"
    'Fill Ki for the unregulated method only
    PacketString = PacketString + Str(Ki) + "***"
    'Fill the data ki characters
    PacketString = PacketString + Left(LocalStr, Ki)
    LocalStr = Right(LocalStr, Len(LocalStr) - Ki)

    'Fill the packet in the correspondign node waiting list
    NodeCol(Sn).Add PacketString
    List1.Item(Sn).AddItem Str(PacketNumber)
Next i

'Fill the last packet in this string
'First fill the packet Number
PacketNumber = PacketNumber + 1
PacketString = Str(PacketNumber) + "***"
'Fill the destination node
PacketString = PacketString + Str(Dn) + "***"
'Fill Ki for the unregulated method only
PacketString = PacketString + Str(Ki) + "***"
'Fill the data ki characters
PacketString = PacketString + LocalStr
'Set the wanted info in the message such as packet no ...

MessageInfo(Val(Label3.Caption)).Add TimeCount
MessageInfo(Val(Label3.Caption)).Add Sn
MessageInfo(Val(Label3.Caption)).Add Dn
MessageInfo(Val(Label3.Caption)).Add PacketNumber
MessageInfo(Val(Label3.Caption)).Add Di
MessageInfo(Val(Label3.Caption)).Add Pi

'Fill the packet in the correspondign node waiting list
NodeCol(Sn).Add PacketString
```

```
    List1.Item(Sn).AddItem Str(PacketNumber)

End Sub

Function GetKi()
    Dim SumB As Long    'The summation of the blocking at each node
    Dim HoldingTime As Long
    Dim Wi As Long  'The number of channels
    Dim Hi As Long  'The real number of flits in a packet
    Dim Ni As Long  ' The number of packets in a message
    Dim Ki As Integer

    Ki = 1

    Do While (1 < 2)

        If (Ki > Ci) Then
            GetKi = -1
            Exit Function
        End If

        HoldingTime = Ki + 2
        Wi = Abs(Dn - Sn)
        SumB = Abs((Dn - Sn)) * (HoldingTime)
        Hi = Ki + 3
        Ni = Ci / Ki

        If ((Ni * (SumB + Wi + Hi - 1)) <= Di) Then
            GetKi = Ki
            Exit Function
        Else
            Ki = Ki + 1
        End If
    Loop

End Function


Private Sub Command7_Click()

    Dim i As Integer
    Dim myString As String
    Dim CharVal As Integer
    Dim strfilter As String
    Dim strFileName As String
    Dim msgCount As Integer
    Dim RetVal As Variant

    'The file management related material
    'strfilter = "StrText (*.str)|*.str|All Files (*.*)|*.*"
    strfilter = "All Files (*.*)|*.*"



    CDlg1.Filter = strfilter
    'Open the common dialog in save mode
    CDlg1.filename = ""
    CDlg1.ShowSave
    'Make sure the retrieved filename is not a blank string
    If CDlg1.filename = "" Then
```

```
    MsgBox "Invalid File Name !"
    Exit Sub
End If

'MsgBox (CDlg1)
'Exit Sub

'If it is not blank, open the file
strFileName = CDlg1.filename


'Open the files for entry
'Open a file for writing strings in it
Open (strFileName + ".str") For Output As #1
'Open a file for writing Pi in it
Open (strFileName + ".pi") For Output As #2
'Open a file for writing Di in it
Open (strFileName + ".di") For Output As #3
'Open a file for writing Sn in it
Open (strFileName + ".sn") For Output As #4
'Open a file for writing Dn in it
Open (strFileName + ".dn") For Output As #5


'Fill the message data in the respective files
Form1.MousePointer = vbHourglass
'TP = 10 'max of di value

NumberOfMessages = Val(Text1.Text)
For msgCount = 1 To NumberOfMessages
    '*****
    'make the string and put it in the file
    Ci = Rnd * Val(Text4.Text)
    If Ci = 0 Then Ci = 1
    'MsgBox Ci, , "Ci"
    myString = ""
    For i = 1 To Ci
        CharVal = (Rnd * 100) + 30 'The +30 is just to skip the special characters
        myString = myString + Chr(CharVal)
    Next i
    'MsgBox myString, , "MyStirng Value is ..."
    'Do the write
    Print #1, myString
    '*****

    'Time between message and another
    Pi = (Rnd * Val(Text3.Text)) 'The + 1 is for the first time it is called to replace the -1
    If Pi = 0 Then Pi = 1
    'MsgBox Pi, , "PI"
    'Do the write
    Print #2, Pi
    '*****

    'Deadline for the message
    Di = Rnd * Val(Text2.Text)
    If Di = 0 Then Di = 1
    'MsgBox Di, , "Di"
    'Do the write
    Print #3, Di
```

```
        'If Di < TP Then
        '   TP = Di
        'End If
        '*****

        Sn = Rnd * MaxNode  'the limit is the MaxNode
        If Sn = 0 Then Sn = 1
        'MsgBox Sn, , "Sn"
        'Do the write
        Print #4, Sn
        '*****

        Dn = Rnd * MaxNode  'the limit is the MaxNode
        If Dn = 0 Then Dn = 1
        'MsgBox Dn, , "Dn"
        'Do the write
        Print #5, Dn
        '*****
    Next msgCount
    Form1.MousePointer = vbDefault


    'MsgBox TP, , "TP"
    'Close the file when completed
    Close #1
    Close #2
    Close #3
    Close #4
    Close #5

End Sub


Sub CheckPackets()
    Dim i As Integer
    Dim LastPacket As Boolean
    Dim PacketNo As Long
    Dim packetdn As Integer
    Dim CurrPacket As String
    Dim SubPacket As String
    Dim StarPos As Integer
    Dim packetKi As Integer




    For i = 1 To MaxNode
        'If i = 1 Then
        '   MsgBox i, , "The IIIIIIII"
        'End If

        'If NodeB(i) = -1 then need to get the blocking time for the next packet if it exists
        If NodeB(i) = -1 Then
            'GetPacketInfo done here
            If NodeCol(i).Count <> 0 Then
                CurrPacket = NodeCol(i).Item(1)
                    'MsgBox CurrPacket, , "Current Packet Value   INITIALIZE"
                StarPos = InStr(CurrPacket, "***")
                PacketNo = Val(Left(CurrPacket, StarPos - 1))
                SubPacket = Right(CurrPacket, Len(CurrPacket) - (StarPos - 1) - 3)
```

```
            StarPos = InStr(SubPacket, "***")
            packetdn = Val(Left(SubPacket, StarPos - 1))
            SubPacket = Right(CurrPacket, Len(SubPacket) - (StarPos - 1) - 3)
            StarPos = InStr(SubPacket, "***")
            packetKi = Val(Left(SubPacket, StarPos - 1))

            NodeB(i) = (Abs(packetdn - i) * (packetKi + 2))
            BVal(i).Caption = NodeB(i)
            BVal(i).Refresh

        End If
    End If



    If NodeB(i) = 0 Then
        'GetPacketInfo done here
        If NodeCol(i).Count <> 0 Then

            CurrPacket = NodeCol(i).Item(1)
                'MsgBox CurrPacket, , "Current Packet Value"
            StarPos = InStr(CurrPacket, "***")
                'MsgBox StarPos - 1
            PacketNo = Val(Left(CurrPacket, StarPos - 1))
                'MsgBox PacketNo, , "packet No in packet"
            SubPacket = Right(CurrPacket, Len(CurrPacket) - (StarPos - 1) - 3)
                'MsgBox SubPacket, , "Sub packet"
            StarPos = InStr(SubPacket, "***")
                'MsgBox StarPos - 1
            packetdn = Val(Left(SubPacket, StarPos - 1))
                'MsgBox packetdn, , "packet destination in packet"
            SubPacket = Right(CurrPacket, Len(SubPacket) - (StarPos - 1) - 3)
                'MsgBox SubPacket, , "Sub packet"
            StarPos = InStr(SubPacket, "***")
                'MsgBox StarPos - 1
            packetKi = Val(Left(SubPacket, StarPos - 1))
                'MsgBox packetKi, , "Packet Ki"


            If (i = packetdn) Then

                CheckLastPacket (PacketNo)
                If MessagePos <> -1 Then
                    LastPacket = True
                Else
                    LastPacket = False
                End If

                If (LastPacket) Then
                    'Set the arrival time in the message info
                    'MsgBox (TimeCount - MessageInfo(MessagePos).Item(1))
                    'MsgBox MessageInfo(MessagePos).Item(5)
                    If ((TimeCount - MessageInfo(MessagePos).Item(1)) <
MessageInfo(MessagePos).Item(5)) Then
                        'messages arrived before deadline
                        Label8.Caption = Str(Val(Label8.Caption) + 1)
                        Label8.Refresh
                    Else
                        'messages arrived after message deadline
```

```
                Label9.Caption = Str(Val(Label9.Caption) + 1)
                Label9.Refresh
              End If


              NodeCol(i).Remove 1
              List1(i).RemoveItem (0)
              List1(i).Refresh
            Else
              NodeCol(i).Remove 1
              List1(i).RemoveItem (0)
              List1(i).Refresh
            End If
            NodeB(i) = NodeB(i) - 1
            BVal(i).Caption = NodeB(i)
            BVal(i).Refresh
          Else
            If (i < packetdn) Then
              NodeCol(i + 1).Add (NodeCol(i).Item(1))
              List1(i + 1).AddItem List1(i).List(0)
              List1(i + 1).Refresh
              NodeCol(i).Remove 1
              List1(i).RemoveItem (0)
              List1(i).Refresh
              NodeB(i) = NodeB(i) - 1
              BVal(i).Caption = NodeB(i)
              BVal(i).Refresh
              If List1(i + 1).ListCount = 1 Then
                i = i + 1
              Else
              End If
            Else
              NodeCol(i - 1).Add (NodeCol(i).Item(1))
              List1(i - 1).AddItem List1(i).List(0)
              List1(i - 1).Refresh
              NodeCol(i).Remove 1
              List1(i).RemoveItem (0)
              List1(i).Refresh
              NodeB(i) = NodeB(i) - 1
              BVal(i).Caption = NodeB(i)
              BVal(i).Refresh
            End If
          End If
        End If
      ElseIf (NodeB(i) > 0) Then
        NodeB(i) = NodeB(i) - 1
        BVal(i).Caption = NodeB(i)
        BVal(i).Refresh

      End If
    Next i
End Sub

Sub CheckLastPacket(pacNo As Long)
  Dim i As Integer

  For i = 1 To Val(Label3.Caption)
    If (MessageInfo(i).Item(4) = pacNo) Then
      MessagePos = i
      Exit Sub
```

```
    End If
  Next i
  MessagePos = -1
End Sub
```

REFERENCES

[1] G. Agrawal, B. Chen, W. Zhao, and S. Davari, "Guaranteeing Synchronous Message Deadline with the Timed Token Protocol," *IEEE Computer*, vol. 43, No.3, March 1994.

[2] C. M. Aras, J.F. Kurose, D.S. Reeves, and H. Schulzrinne, "Real-Time Communication in Packet-Switched Netwoks," *IEEE Proceeding*, vol. 82, pp. 122-139, Jan 1994.

[3] B. Chen, H. Li, and W. Zhao, "Meeting Delay Requirements in Computer Networks with Wormhole Routing."

[4] P. H. Enslow (ed.), *Multiprocessors & Parallel Processing*, Wiley, New York, 1974.

[5] S. Felperin, P. Raghavan, E. Upfal, "A theory of Wormhole routing in Parallel Computers," in *Proc. 33rd Annual Symposium on Foundations of Computer Science*, pp.563-572, 1992.

[6] T. Y. Feng, "A survey of Interconnection Networks," *IEEE Computer*, 14(12): 12-27, 1981.

[7] A. J. van de Goor, *Computer Architecture & Design*, Addison-Wesly, Reading, MA, 1989.

[8] K. Hwang, *Advanced Computer Architecture,* McGraw-Hill Book Co., 1993.

[9] L. Johnson, "Communication in Network Architectures," in Suaya and Birtwistle (eds.), *VLSI And Parallel Computation*, Morgan Kaufman, San Mateo, CA, 1990.

[10] B. H. Kwan, "Intelligent Resource Allocation in High Speed Networks Using Wormhole Routing," 1997.

[11] J. P. Li, M. W. Mutka, "Priority Based Real-Time Communication for Large Scale Wormhole Networks," *Parallel Processing Symposium*, 1994.

[12] X. Lin, P. K. McKinley, and L. M. Ni, "Performance Evaluation of Multicast Wormhole Routing in 2D-Mesh Multicomputers," *Proc. Int. Conf. Parallel Processing*, vol. I, pp. 435-442, 1991.