A Frame-based Arbitration and Scheduling Technique for Multiprocessor Video-on-Demand Systems

by
Smitha Srinivasan

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
Canada

September 2005

Your file Votre référence
ISBN: 0-494-10121-0
Our file Notre référence
ISBN: 0-494-10121-0

Abstract

Multimedia services such as Video-on-Demand (VoD) are gaining in popularity. Video data is time-constrained, bursty and requires massive storage space. Multiprocessors make good VoD servers since they can handle a large number of concurrent requests. Traditional multiprocessors use general-purpose network service algorithms that are not conducive for VoD. Simple fairness algorithms such as Round-Robin, provide no service guarantees and cannot manage the congestion caused by burstiness of data. Algorithms that provide network service guarantees such as Stop-and-Go, cannot manage the size of video data. Stop-and-Go's admission control policy leads to under-utilization of the network, poor throughput and excess total latency for video data.

We introduce a new algorithm called Frame-based Arbitration and Scheduling Technique (FAST), which is designed for handling video traffic. We show that FAST's policy of allowing bursty data into the network leads to superior throughput and total latency. The low queuing delay and packet loss figures also emphasize that FAST is suitable for use in multiprocessor-based VoD servers.

Acknowledgments

I would like to thank Dr. Sivarama Dandamudi and Dr. Govindan Ravindran for their guidance and advise during the course of my research. The meetings we had and the many well-placed pointers I received from them, helped me narrow down the research area very quickly. Their patience and understanding despite the many delays I faced, is to be appreciated.

I am indebted to my parents for giving me a solid education and providing a caring, supportive and fun environment for me during my formative years. Their aid during my times of need in the past few years was invaluable and much needed. My brother Anand deserves special mention for giving me his laptop so I could work from anywhere and for bullying me about my progress with respect to the dissertation.

My husband Anand was the source of inspiration that led me to complete my dissertation. I drew on his technical expertise in the area of telecommunications and performance analysis, at many points when I was stuck in my research. His unique sense of humor kept my spirits up. His support in the form of well-timed lectures kept me from giving up hope of completing this dissertation.

My children Aditi and Anika deserve special mention for being cooperative and allowing me to do my work unhindered. Their patience and understanding even at such young ages, is to be appreciated. I would like to thank my neighbor and friend Toopana for baby-sitting my children on numerous occasions, and giving me time to work on this dissertation.

Finally, I would like to thank my employers – Mitel Networks and Nortel Networks for their financial support in the form of educational assistance for completing this program.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ABR	Available Bit Rate
ACK	ACKnowledgement
ATM	Asynchronous Transfer Mode
bps	bits per second
CBR	Continuous Bit Rate
DVD	Digital Video Disc
DSL	Digital Subscriber Line
DVC	Dynamic Virtual Channel
FAST	Frame-based Arbitration and Scheduling Technique
FCFS	First-Come First-Served
FIFO	First-In First-Out
IP	Internet Protocol
ISN	Integrated Services Network
MPEG	Moving Picture Experts Group
NACK	Negative ACKnowledgement
NIC	Network Interface Controller
PM	Processor-memory Module
QoS	Quality of Service
RCQ	Rotating Combined Queuing
RR	Round-Robin
SG	Stop-and-Go
SVC	Static Virtual Channel
TE	Traffic Engineering
VBR	Variable Bit Rate
VCR	Video Cassette Recorder
VoD	Video-on-Demand

Chapter 1

Introduction

Over the last few decades network performance has been drastically improved due to advances in network architectures, theories and hardware. Early on, networks were mostly telephony oriented while there were some enterprise networks. Not much data was carried on these networks due to bandwidth constraints. With the introduction of ATM, the backbones provided mechanisms to address predictability. Bandwidth and latency problems were also alleviated. With the introduction of IP, sending voice, video and data predictably on the same network became easier. This paved the way for Integrated Services Networks, which carry a wide range of traffic types and enable voice, video and data convergence.

Predictability allows for traffic engineering of the data. Traffic engineering allows real-time data transfer and quality of service. Quality of service and traffic engineering have made it possible to efficiently use data networks to address real-time transfer of data. Enabling real-time transfer of data implies sending video packets over these networks. Real-time video requires packets to arrive within their deadline. If the deadline is missed, then the viewer is subjected to unacceptable jitter while watching the video. In this manner, video traffic, which requires minimum worst-case delay, differs from traditional

data traffic that requires good average performance. This problem is further complicated by the fact that video traffic is much bulkier than regular data traffic due to its size.

To address the minimum worst-case delay problems faced by video traffic, video servers that store, retrieve and transmit video efficiently were introduced. These servers handle the video packets that must be played in a real-time manner and must be received by the client in time for continuous playback. Many advances have been made in researching the architecture of such systems also referred to as Video-on-Demand servers. In general it has been found that parallel computers, multiprocessors and clusters of workstations work better due to their parallel nature when handling such bulk transfer of data.

There is a big industry that requires efficient Video-on-Demand servers. Some of the players in this field are companies that offer video streaming services, satellite television and cable companies, and integrated service providers that offer on-line services.

1.1 Players in the Video Streaming Market

Movies on demand, video kiosks, distance learning, computer-aided training, video library, tele-medicine, remote monitor and control, home shopping, and entertainment are some of the application areas requiring efficient Video-on-Demand servers. Industries providing some of these services are mentioned below.

Traditionally phone companies such as Bell Canada were only interested in the market for voice switching and providing voice services. But with the deregulation of local and

long-distance services by governments, more and more players are encroaching upon their territory by providing cheaper local and long-distance services. Even cable companies have started to offer voice services over their pre-installed bases of co-axial cables. In order to diversify and to compete with such companies, the big phone companies have moved into services such as providing internet connectivity over DSL and into television and movie broadcasting. These big phone companies provide video services by using satellites and dish antennae installed at individual homes. There are also new proposals from incumbent telephone companies to provide video broadcasting over DSL. By branching into the video broadcasting industry, these companies now face some very interesting problems related to video streaming.

The second type of players in the movie broadcasting and video streaming business are cable television companies such as Shaw Communications. These companies, which have been in the video broadcasting business for decades have a large installed base of co-axial cables running to individual homes. Television channel programming is broadcast to all subscribers regularly over these cables. However, with the increase in competition and advances in video streaming, these companies are looking to carve a niche for themselves in the real-time Video-on-Demand industry.

The third type of players in the video broadcasting business are video and movie rental libraries such as Rogers. Such companies rent out movie tapes and DVDs to individual customers. With this approach, the customer has to physically go to the store, rent the video and return it when they've finished watching it. To alleviate the need for two trips

to the store and to entice the many customers who would want to watch videos from the comfort of their homes by just making an on-line request for a particular video, these libraries are branching into the Video-on-Demand streaming industry.

For all of the types of companies mentioned, once Video-on-Demand comes into the picture, they face all the problems related to network architectures and real-time video streaming. The following sections mention some of the generic requirements and problems in such a scenario.

1.2 Video-on-Demand Systems

The multimedia industry desperately requires Video-on-Demand systems capable of delivering a wide variety of movies and video clips concurrently to thousands of subscribers' televisions.

Multimedia servers process user requests for images, audio and video streams. These data types pose a challenge to the research community since they require significant storage space and guaranteed services in order to reach the requesting node on time. Video data in particular has rigid time constraints, since a video frame has to reach before the previous frame finishes playing, in order to avoid jitter.

A Video-on-Demand system is a repository of videos that retrieves and plays high quality videos in real-time. The system must ensure that the video is delivered to the client in time for continuous playback. Many types of architectures have been proposed for Video-

on-Demand servers, including single server and parallel server architectures. Due to scalability and fault tolerance problems with single server architectures [14], parallel server architectures for Video-on-Demand servers become an attractive alternative. A parallel video server is a multiprocessor consisting of multiple nodes where each node is a Processor-Memory module with independent disk drives. The nodes are located close together and are connected to each other via a parallel interconnect.

1.3 Integrated Services Network

Historically, multiprocessor interconnection networks had to deal with two major types of traffic – Read and Write data. Read packets were meant to read from a memory location, while write packets were meant to update a memory location. Multimedia traffic, on the other hand, included images, audio streams and video streams. Storage and retrieval of multimedia data is constantly gaining importance due to the problems posed by the large storage and high bandwidth requirements of such data. Of all the data types, digital video is the most challenging since it has to be retrieved in time for continuous playback.

1.3.1 Traffic Types in Integrated Services Networks

In today's data network, real-time multimedia and traditional traffic types co-exist. Traffic types such as high-quality video images, audio streams and regular data communication packets, all vie for a common set of network resources. An integrated services network provides the capability to support various traffic types in a manner such that the service guarantees to a particular traffic type are not affected by the presence of the other traffic types.

When multiprocessors are used as multimedia servers the following traffic types compete for the interconnection network resources:

- Available Bit Rate (ABR): Parallel computations where there is no fixed deadline but packets must do their best to reach sufficiently in time. Also called "best-effort traffic", these packets do not require any service guarantees, except a minimum amount of bandwidth.
- Constant Bit Rate (CBR): Uncompressed video and audio traffic where the packets have a deadline before which they must reach the requesting node. Also called "continuous traffic" because traffic is characterized by a fixed arrival rate. These packets require rigid service guarantees of bandwidth and fixed delay bounds.
- Variable Bit Rate (VBR): Compressed video (MPEG) and audio traffic with a fixed deadline before which packets must arrive at the requesting node. Such traffic do not have a fixed arrival rate and can arrive in bursts. Usually, VBR traffic is bursty with varying amount of each burst. In spite of the bursty nature of the traffic, certain service guarantees for bandwidth and minimum delay bounds are essential. Due to the unpredictable nature of VBR traffic, networks find it difficult to efficiently support this type of traffic. Currently, however, VBR is the most popular type of traffic found on networks, due to the growth of compression techniques.

1.4 Challenges Faced by Multiprocessor Video-on-Demand Servers

Multiprocessor Video-on-Demand servers transfer video data over an interconnection network. Main issues in such systems include large storage requirements, continuous streaming of video and high network bandwidth required for video playback.

1.4.1 Video Storage

Typically video data such as a movie is very large. For example, a 90 minute uncompressed movie requires approximately 150GB of storage. A server that stores movies and plays them on demand must have access to a large movie repository. This implies a requirement of several hundred gigabytes of internal and external disk space. Compressing movies stored on disk alleviates this problem a little [16, 26]. MPEG-1 [30] provides a compression rate of 1:25. MPEG-4 [30] and H.264/AVC [36] are the newest and most advanced compression techniques.

To minimize video data retrieval delay, compressed video is divided into blocks and striped across several disks connected to several processors in the video server.

1.4.2 Scalability

After a Video-on-Demand system is built, the videos will probably be upgraded over time, new videos may be added, old videos may be removed and nodes may be removed for maintenance. To scale a multiprocessor Video-on-Demand server, new processors are added in order to handle the extra load created by storage of new videos. The design of a

Video-on-Demand server should be flexible enough to allow changes easily without severely interrupting its regular service.

1.4.3 Load Balancing

All the processors in a multiprocessor Video-on-Demand server should share the load equally. Overloading of some nodes while others being under utilized leads to load imbalance and performance problems.

Many customers require access to popular videos frequently. It is likely that the processors or disks responsible for these popular videos are overloaded and too busy to handle requests while those storing less popular videos are idle. This is known as the hotspot problem of popular videos. A good Video-on-Demand server architecture should balance the load properly across all processors and disks thus eliminating the hotspot problem.

1.4.4 Fault Tolerance

In a multiprocessor Video-on-Demand server, some processors may fail over time due to hardware or software problems. In order to maintain guaranteed service for all the clients, upon a failure, requests assigned to the failed node should be handled by the other active nodes without severely affecting the playback to other clients.

1.4.5 Providing Full VCR-like Functionality

In a true Video-on-Demand system, the user has complete control during video playback with support for full-function virtual VCR capabilities [28], including fast forward, rewind, pause and random positioning.

1.4.6 Constant Stream Rate

Video streaming [16] is a technique for transferring video data such that it can be processed as a steady and continuous stream. Reference [26] defines video streaming as - real-time transmission of stored video. Streaming technologies are becoming increasingly important with the growth of the Internet because most users do not have fast enough access to download large multimedia files quickly [27]. With streaming, the client browser or plug-in can start displaying the data before the entire file has been downloaded. Currently, Video-on-Demand systems are being designed for streaming compressed (MPEG-4 [30]) videos over the Internet [29].

When a movie is played to a client, the stream rate is constant. That is, there should be no delay between frames and the client must not see any jitter or disturbance in picture. Typically, a continuous playback, at a rate of 1.5 Mbps is required. This implies that when a movie block B1 that has arrived at a client starts to play, the next movie block B2 must have arrived, been buffered and must be ready to play before block B1 finishes playing. This introduces some latency during video playback. Methods are being devised to decrease this latency while ensuring continuous playback of the video [35].

1.4.7 Network Bandwidth

Striping divides the video into blocks for convenient storage and retrieval. Video blocks are typically large – of the order of kilobytes and megabytes. These blocks have to travel over the interconnection network to reach the requesting node. The link widths of multiprocessor interconnection networks are small – 64, 128 or 256 bits. This bandwidth is not enough for transporting an entire movie block efficiently over the interconnection network. Also, intermediate switches or Network Interface Controllers (NIC) usually do not have enough buffer space for storing such a large amount of data. We propose using wormhole routing to divide packets containing a block of the video into smaller, more manageable-sized units called flits. If the size of the flit is the same as the link width, then every flit can be transferred over a link in a single system clock cycle.

1.4.8 Versatile Functionality of the Network Interface Controllers

Video data not only has a fixed deadline but it can also be bursty. Presently multiprocessor switches or Network Interface Controllers (NIC) are designed to be simple to support traditional data using simple arbitration schemes. These link arbitration schemes such as Round-Robin and First-Come-First-Served (FCFS) provide resources uniformly to all connections without providing any Quality of Service (QoS) guarantees for real-time connections requiring continuous playback. For providing such guaranteed bandwidth for connections, the NIC or switch should be able to allocate link bandwidth and buffer space on demand and should also be able to schedule data efficiently [13]. At the same time, the NIC should be able to fairly support traditional data connections. Hence we have to make the NICs more flexible and intelligent.

1.5 Areas Addressed by this Thesis

This thesis attempts to solve the following critical issues associated to multiprocessor Video-on-Demand servers, specifically, those that relate to the multiprocessor interconnection network architecture:

- Interconnection network topology, video distribution among nodes, and switching and routing techniques.
- Queuing and scheduling at NICs to improve worst-case delay guarantees and assure constant stream rate.
- Using virtual channels to improve the intelligence and flexibility of NICs so that they can efficiently support both video and traditional data types.

1.6 Thesis Contributions

This thesis identifies main problems facing real-time multiprocessor-based Video-on-Demand servers and proposes solutions for these problems. A comprehensive architecture of a Video-on-Demand server with special emphasis on network and communications issues is presented. System design in terms of network topology, network elements, switching, routing, flow control, NIC design and scheduling algorithms at NICs is explained in detail. An efficient and unique queuing and scheduling algorithm at the NIC called Frame-based Arbitration and Scheduling Technique (FAST) is proposed and its performance is compared to the Round-Robin and Stop-and-Go algorithms. It is shown that the performance of FAST in Video-on-Demand servers is superior to Round-Robin and Stop-and-Go.

1.7 Thesis Organization

Chapter 2 provides an overview of various Video-on-Demand server architectures. Existing approaches for enhancing the performance of the interconnection network of a multiprocessor Video-on-Demand server are explained and the challenges and issues are highlighted.

Chapter 3 explains the architecture of the proposed server, specifically the network and communications mechanisms of the proposed system. Details about the simulator used to gather the performance parameters are also provided in this chapter.

Chapter 4 explains the proposed FAST algorithm. FAST queuing and scheduling are explained and implementation details are provided.

Chapter 5 presents the simulation results obtained from performance analysis of the proposed algorithm. The input traffic patterns used for simulation purposes are provided and performance metrics explained.

Chapter 6 concludes this thesis by summarizing the major contributions of this dissertation and, providing directions for future work.

Chapter 2

Background

A number of processors networked together and working in parallel is called a multiprocessor. Multiprocessors are normally used for their speed and ability to solve large problems. Many application areas require the computational power of multiprocessors. Some of these are - geographical information analysis, weather forecasting, nuclear calculations and computations in the field of astronomy.

Researchers have been studying multiprocessors for some time now, so as to improve the speed and the scalability of these servers. Some research areas in this field are – study of the topology of the network that connects the processors together (the interconnection network) [31, 32], architectural issues of the multiprocessor as a whole [33], and memory management issues [34].

In a parallel Video-on-Demand server the following factors play major roles:

- Internal architecture of the server,
- Scheduling of movie requests,
- Storage and retrieval of video blocks, and,
- Performance of the interconnection network.

2.1 Video Server Architectures

There has been a lot of research on architectures that would best suit a Video-on-Demand server [17, 18, 19]. While these works focus on the storage characteristics and optimal scheduling of multimedia requests, not much work was done on the performance of the interconnection network.

2.1.1 Single Server Architecture

Single server Video-on-Demand servers as the name implies, have only one server that handles requests for videos. This server could be a desktop computer or it could be a massively parallel computer. Entire videos are stored on one or more shared disks with little to no redundancy built-in. Such architectures are not scalable and lack fault tolerance. Due to these limitations, single server architecture has been found to be inadequate for large scale, highly available Video-on-Demand servers [6, 14].

In reference [19], the limitation of single server architecture and the reasons for moving to a multiple server architecture are explained. With the increase in demand for video quality and services, the I/O limitations and the single point of failure are becoming major obstacles in the design of the video server. Along-with the tendency of substituting expensive and high computation machines with cheaper processors, Video-on-Demand server design is in favour of a multi-server design. Such systems are designed to improve the scalability and performance over a single-server design.

2.1.2 Parallel Server Architectures

A parallel Video-on-Demand server consists of multiple Processor-Memory modules and multiple storage disks connected together by a parallel interconnect.

Reference [18] defines three kinds of nodes in parallel video servers; *storage nodes*, *delivery nodes* and *control nodes* all connected together by an interconnection network using switches or shared hubs. The *storage node* is responsible for storing video clips, retrieving requested data blocks and sending the blocks to delivery nodes. Partitioned video blocks are striped between storage nodes in a Round-Robin fashion. The *delivery node*, on the other hand, is responsible for taking requests from clients and forwarding these requests to the control node for scheduling. Responses in the form of video blocks sent by storage nodes are buffered in delivery nodes, where data is re-sequenced if necessary, and then forwarded to clients. The *control node* is responsible for admission control, network scheduling of requests, and, video content management. Upon receiving requests from delivery nodes, the control node makes decisions about whether to admit the request or not and then schedules the request if admitted.

The following sections outline architectures for parallel Video-on-Demand servers using the 3 types of nodes (storage, delivery and control nodes) mentioned above.

Shared-Memory Multiprocessors

In shared-memory multiprocessors, a set of storage nodes and a set of control nodes are connected to a shared memory. Video data is sent to the memory buffer through a high-

speed bus and then on to clients. A mass storage device handles the capacity required for servicing multiple requests. However, this model has not proven to be scalable.

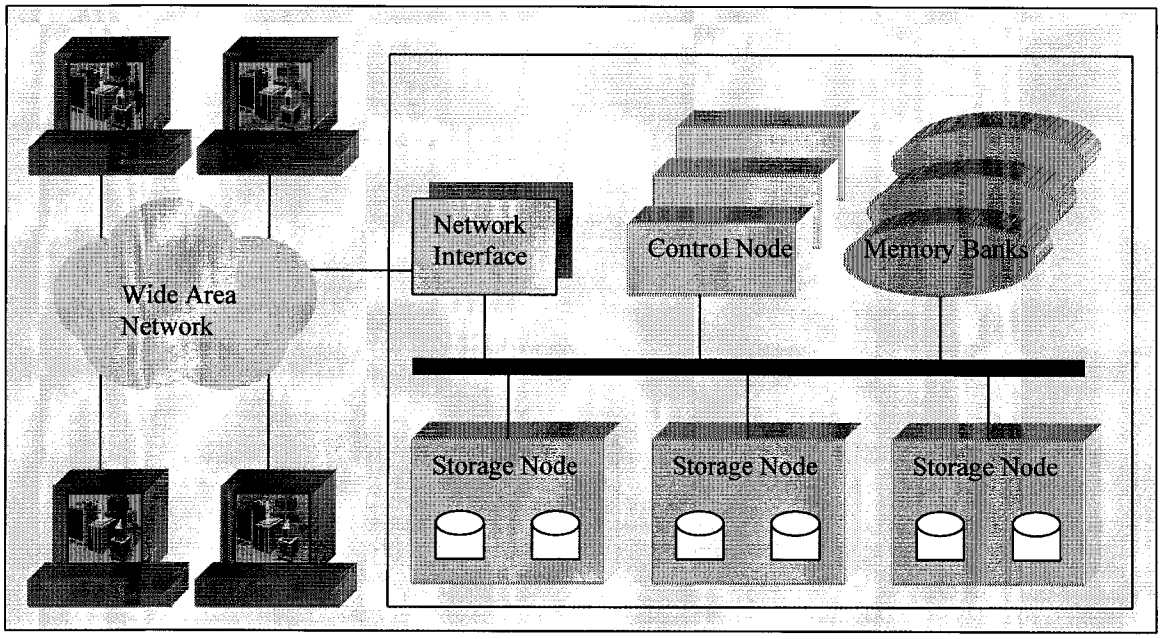


Figure 2.1: Shared-memory multiprocessor architecture

Distributed-Memory Clustered Systems

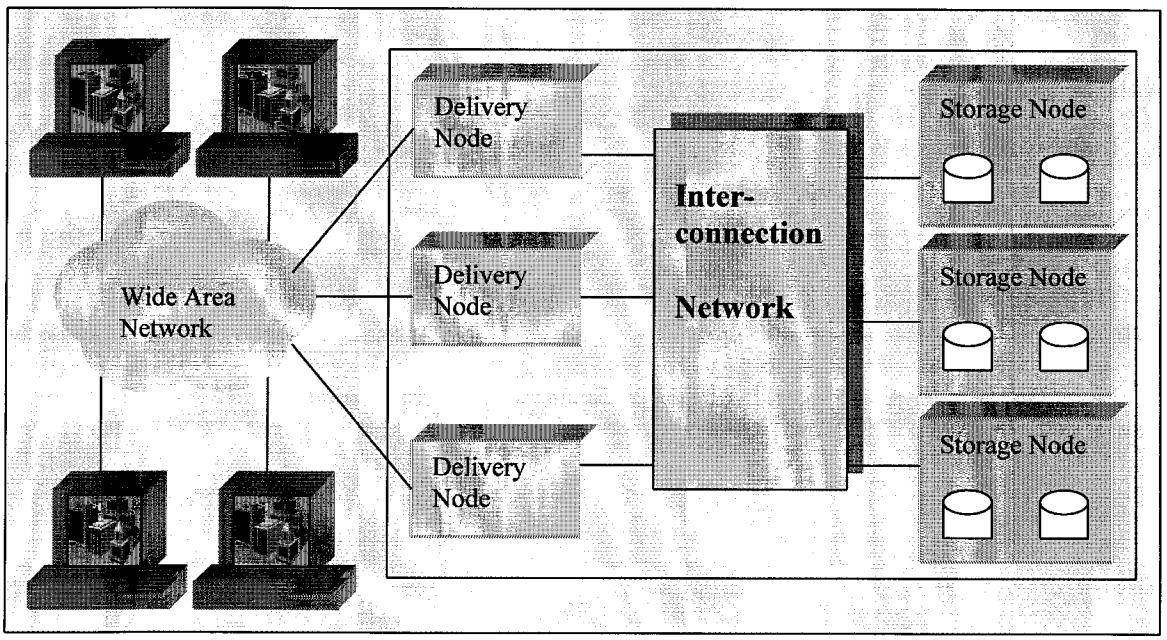


Figure 2.2: Distributed-memory clustered system architecture

In distributed-memory clustered systems [17, 22, 23], a set of storage nodes and a set of delivery nodes are connected by an interconnection network. Data is retrieved from storage nodes and sent to delivery nodes, which send it on to the clients. This type of architecture is easy to scale.

Flat Architecture

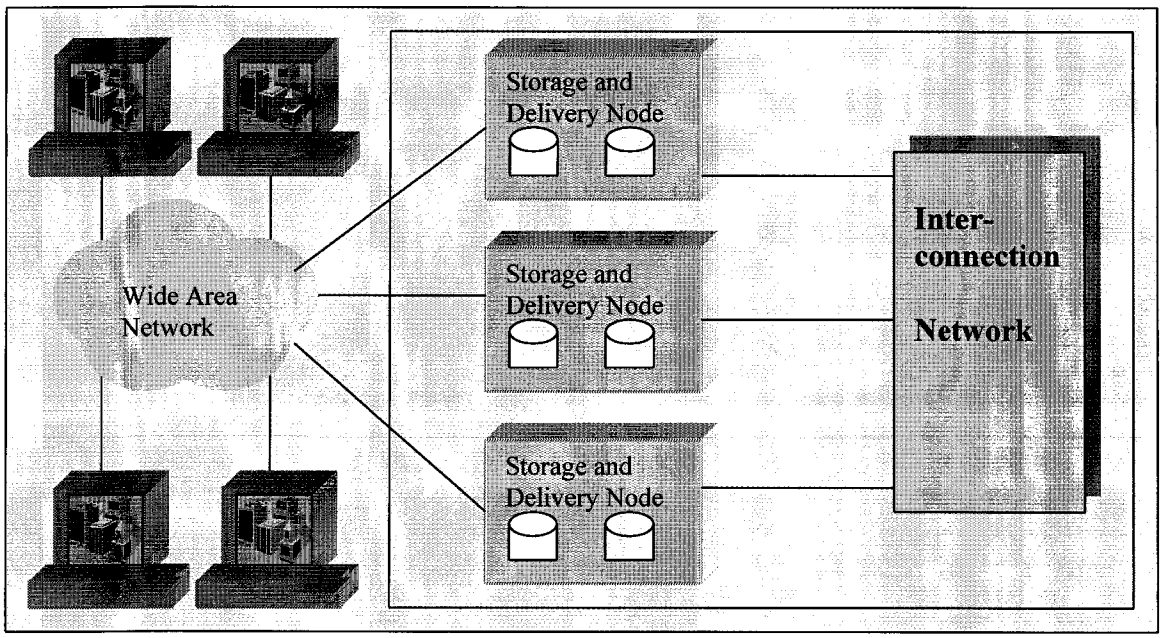


Figure 2.3: Flat architecture

In a flat architecture, a node acts as both a storage node and a delivery node. Since in [17], this flat architecture is suggested to produce better network utilization, we use this model in our research. In a flat architecture a storage node may transmit video data to a delivery node residing in the same processing node. In this case, data is transferred within the processing node without utilizing the interconnection network bandwidth. This provides surplus network bandwidth which can be used for performing aperiodic requests such as adding video files.

Direct Access Systems

Direct access architecture [24, 25] eliminates the need for delivery nodes used in clustered architecture, by connecting storage nodes directly to the network via a network interface. Although this is a simple architecture, the order of arrival of blocks is not guaranteed.

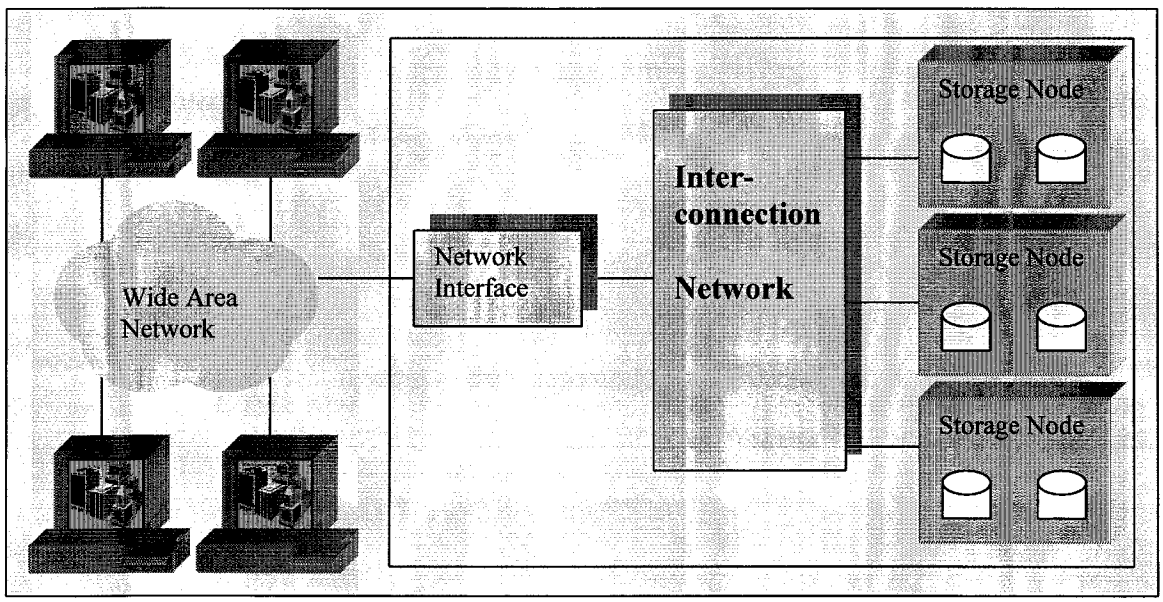


Figure 2.4: Direct access system architecture

2.2 Major Factors Affecting Parallel Video-on-Demand Server Design

When designing a multiprocessor Video-on-Demand server, the following factors should be considered.

2.2.1 Data Distribution across Multiple Storage Nodes

Parallel Video-on-Demand servers have the video data striped across multiple servers [14]. For this study, we consider parallel Video-on-Demand servers with the video data striped uniformly across all servers. Striping is a method for distributing data over

multiple devices in order to improve capacity and reliability. It ensures fault tolerance and load balancing, since the entire video is not stored on a single storage medium.

Striping can be further classified as either time striping or space striping. In the time striping technique, a video file is striped or divided in units of frames across multiple nodes. Hence, time-striping divides a video in terms of constant playback time among several nodes. The granularity can be as small as one frame-time. Time-striping simplifies video streaming since all stripes play for the same amount of time. On the other hand, in the space striping technique, the video file is divided into segments of fixed size, i.e. fixed amount of storage space. Space-striping simplifies storage and buffer management at nodes since all stripes are the same size. More information on striping can be found in [14, 19].

2.2.2 Reordering of Received Data

Since the data is striped and distributed across multiple servers, the stripes must first be put in order and then assembled into a single stream before delivering the video to the client. The unit that is responsible for reordering and merging the stripes is called a proxy. The proxy can be implemented either in hardware or in software in one of the following ways [14] –

1. Proxy-at-server, where the proxy is implemented at the server itself,
2. Independent proxy, where the proxy is implemented at an independent computer, and,
3. Proxy-at-client, where the proxy is implemented at the client.

2.2.3 Interconnection Network Topology

The topology of the interconnection network [20] plays a big role in contributing to the speed of a parallel system. Usually regularly connected networks where there are multiple paths from a source node to a destination node, are preferred. At the same time the number and length of wires being used to make these connections should be as low as possible. Examples of good interconnection network topologies are Mesh, Torus, Hypercube, N-Cube etc. (See Figure 2.5).

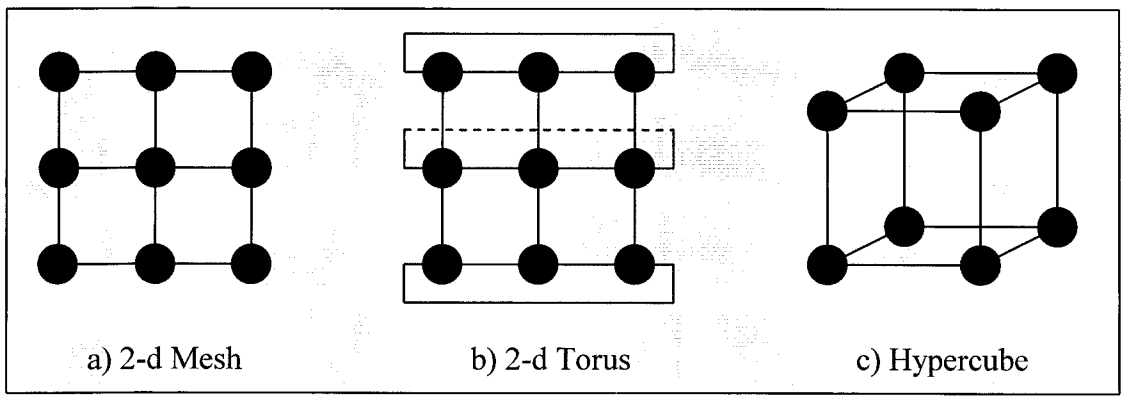


Figure 2.5: Popular network topologies

The nodes in the multiprocessor perform tasks in parallel and communicate via the interconnection network by passing messages. The time spent by a message in the network (latency) has to be minimized, thus increasing throughput. Effective switching, routing, queuing and scheduling techniques are employed to achieve this goal.

2.2.4 Switching

Switching in multiprocessor interconnection networks [20] deals with forwarding of packets from one network link to another. To achieve this, network resources such as

channels and buffers are allocated to packets as they travel through the network. The switching element in the interconnection network is referred to as the Network Interface Controller (NIC). Each switching element has input channels, and output channels, as shown in Figure 2.6. The channels carry messages from different sources bound for various destinations. The switch looks at information about the destination carried by the message in an input channel and decides which output channel the message must be routed to. The switch or NIC is also linked to its local Processor-Memory (PM) module by means of local input and output channels.

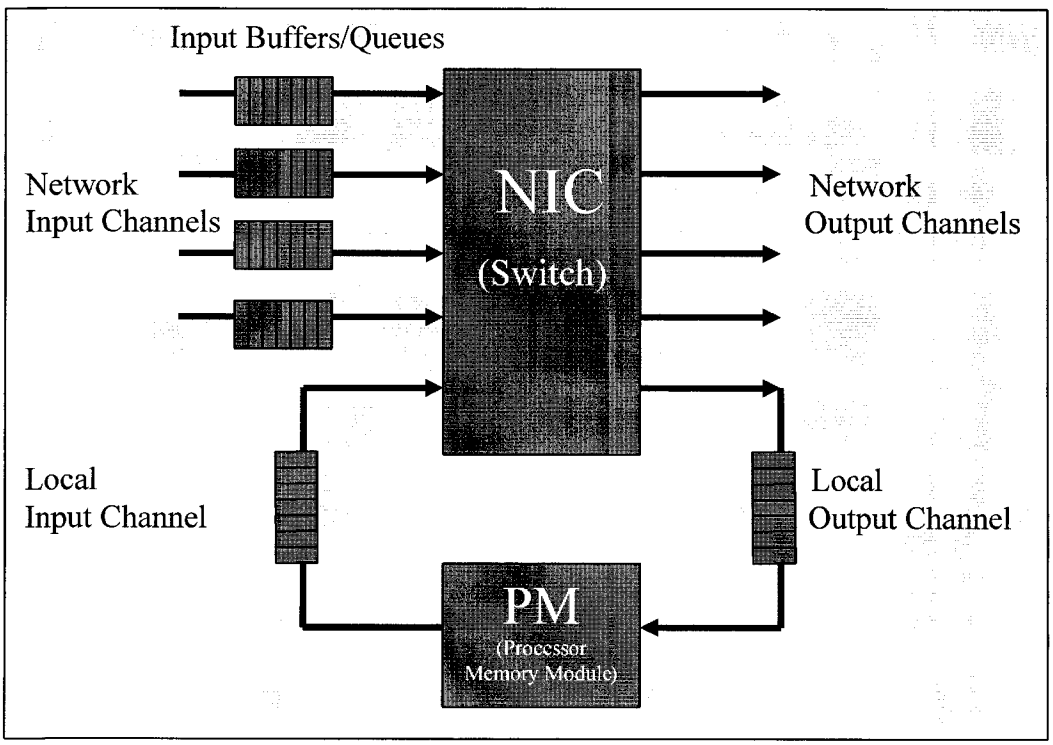


Figure 2.6: Network interface controller

Each channel has some buffers associated with it. The buffers usually arranged into queues - store messages or packets that are in transit between links. In the model we are considering, only the input channels have buffers for storing received packets. When a packet arrives at a switch, it is stored in the buffer queues until it can be forwarded to the

desired output channel. The packet travels over this output channel and arrives at the next switch.

Various switching techniques such as store-and-forward switching, cut-through switching and wormhole routing are used, to forward a packet from one link to the next. The store-and-forward method switches the message as a whole, storing it in intermediate buffers, if an output channel is not available. It has the disadvantage of requiring large buffer sizes at intermediate nodes to store an entire message. Cut-through switching provides a few improvements over store-and-forward switching. In cut-through switching when a packet header arrives at a node, it is forwarded towards its destination without necessarily waiting for the entire packet to arrive. The header reserves a path as it traverses the links from source to destination, the packet bodies follow the path and the packet trailer releases the links after it passes through. Wormhole routing is a variation of cut-through switching. In wormhole routing, packets are divided into smaller units called flits and flits are then switched from node to node in a pipelined fashion. Since wormhole routing is the switching solution chosen for our implementation, we discuss wormhole routing further in the next section.

2.2.5 Wormhole Routing

In wormhole routing [1, 2, 3, 4, 5], a packet is divided into small units called flits. The first flit is called the header flit and contains routing information. The header flit establishes the path from source to destination, while the other flits just follow the header. The packet thus moves from node to node like a worm and packets are not allowed to

interfere with each other (see Figure 2.7). Flits from different packets are not allowed to mix, as shown in Figure 2.7. Once the header starts moving, the other flits follow it in succession as only the header flit carries routing information.

Wormhole routing is a good alternative to Store-and-Forward switching due to its low latency and requirement for small buffers. However, there are potential issues of deadlock and starvation associated with wormhole routing. For example, when the header gets blocked, the worm may span several nodes and might block other packets. If the header is waiting for a channel blocked by another packet and that packet is itself waiting for this worm to free the resources, then a deadlock situation may arise.

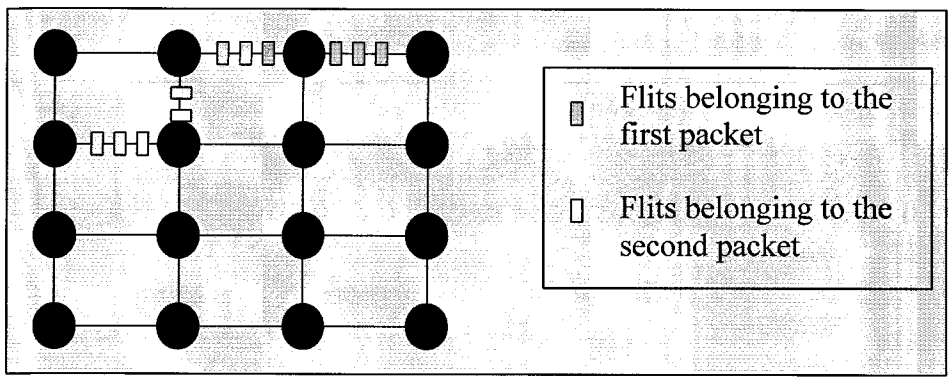


Figure 2.7: Wormhole routing

2.2.6 Virtual Channels

Flow control deals with two types of network resources - channels and buffers. Typically, a set of buffers is assigned to each physical channel.

The buffers belonging to a single physical channel are divided into a number of queues - one queue for each virtual channel [6, 7]. Virtual channels can hold one or more flits of a

packet. If there had been only a single queue associated to each physical channel, a blocked flit on the head of this queue would block every other packet wanting to use this channel. If instead, the physical channel were divided into a number of virtual channels, each with it's own queue, then the blocked header would not block packets in other queues from using the channel, while it is waiting for an output channel to become free. This technique solves deadlock issues faced by wormhole routing. The number of virtual channels into which a physical channel should be divided could be fixed – "static" or could be determined based on traffic conditions – "dynamic".

The existence of virtual channels allow for scheduling strategies to be used for routing packets based on priority, age etc. Although the extra control logic is an overhead, the advantages of making effective use of both the channel and buffer, far outweighs this disadvantage.

2.2.7 Routing

Routing in a multiprocessor network refers to the process by which a packet is transmitted from its source node to a destination node. The path taken by the packet to reach from its source to its destination is referred to as the 'route'. The nodes along the packet's route are called hops.

Direct multiprocessor networks have simple, predictable topologies. This allows us to use algorithmic routing as opposed to table-based routing. In distributed algorithmic routing a

packet's next stop is determined at each NIC when the packet reaches there, by examining the packet's final destination.

Distributed routing can be classified into deterministic and adaptive routing. Deterministic routing is done based solely on the packet's source and destination nodes. Adaptive routing techniques take advantage of the fact that there may be multiple routes to a destination. The packet is forwarded on a link after accounting for several factors such as the current network condition and the number of hops to the destination.

2.3 Existing Research

While a lot of work has gone into researching parallel Video-on-Demand servers, this thesis concentrates on the design of the interconnection network in order to make it fast and reliable. Performance of a Video-on-Demand server is greatly affected by factors such as interconnection network topology, switching mechanisms, flow control policies, routing, NIC design, queuing and scheduling methods.

A fair amount of research has gone into developing efficient algorithms to meet the performance requirements of real-time applications in general-purpose and multiprocessor networks. Most of the existing techniques can be classified into either priority-based methods or frame-based methods. Priority-based techniques perform link arbitration at the NICs based on priorities assigned to packets. However, most of these techniques are unfair to lower priority traffic. In frame-based techniques, the time axis is divided into frames and connections reserve slots in each frame based on average rate of

transmission. A strict admission control policy is imposed on source nodes. One of the most common negative effects of using this technique is bandwidth wastage due to unused time slots.

Some scheduling policies such as Virtual Clock, Stop-and-Go and Rotating Combined Queuing [13] do not explicitly use message deadlines but just attempt to keep the worst-case message delay small and bounded.

In the following sections we discuss these algorithms in detail.

2.3.1 Virtual Clock

Virtual clock [8] is a priority-based method that assigns each connection a virtual clock value based on the specified Average Rate (AR) of transmission. The Virtual Clock (VC) ticks at each packet's arrival to the network by a value of VTick = 1/ AR. The packet is stamped with this VC value and the switch transmits packets in the order of increasing VC values. Priorities can also be implemented by decreasing every flow's VC value by an amount P representing priority, at the start of the flow. Flow control is non-blocking, since the last packet in the queue is dropped if buffer space is less.

If a connection sends packets according to AR, VC of that connection is in the vicinity of the real time. If VC >> real time, this implies that the connection is being bursty and feedback is sent to the source node of bursty connections in order to ask them to reduce their packet input into the network. In this way, admission control is maintained. Credit

accumulation is not allowed, so connections that transmit at lower rates than the specified average cannot utilize the unused time slots in future by being bursty. Hence burstiness is not supported under any circumstance. This implies that some network resources remain unused and cannot be used efficiently.

Virtual Clock is a simple method as long as the number of connections is less. As the number of connections increases, the complexity at the switch increases proportionally. Virtual Clock facilitates priority implementation and independence between connections. However, penalizing bursty flows may cause bandwidth wastage if some flows are transmitting at a lower rate than that specified. Some other effects of using virtual clock are – packets get dropped and it is difficult to analyze the delays associated to a connection since a particular connection's delay is associated to priorities of other connections. Virtual clock is a work-conserving algorithm and cannot provide deterministic delay bounds. Apart from the above virtual clock has been implemented for long distance packet switching networks and does not fit very well in a multiprocessor scenario.

2.3.2 Stop-and-Go

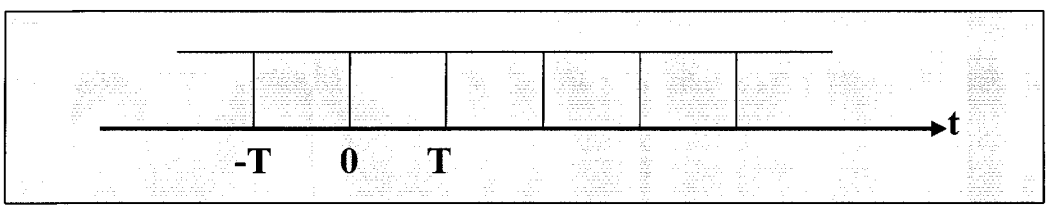


Figure 2.8: Time frames

Stop-and-Go [9, 10] is a frame-based approach where the time axis is divided into fixed frames of size T, as shown in Figure 2.8. Every connection specifies the average rate of transmission 'r', used to reserve slots in the frame. A strict admission control policy imposed at the source nodes disallows connections from using more than the reserved bandwidth or time slots.

The method works on a principle of smoothness that states that every connection is (r, T) smooth, when during a frame of size T, arrived packets from a particular connection collectively have r*T bits. Stop-and-Go queuing at switches preserves the initial smoothness property. All packets that arrive at the input links of a switch during a particular frame are considered for scheduling to the output links at the same time. Hence all packets that arrive during the same time frame are treated equally. Stop-and-Go specifies that the transmission of arriving packets should be delayed until the beginning of the next departing frame in order to guarantee the smoothness property.

Using the Stop-and-Go method of delay calculation, every connection's delay is guaranteed to be bound between (H*T, 2*H*T), where H is the number of hops traveled by the packets of a particular connection and T is the frame size. This constant delay value, which can be kept reasonably low by keeping the frame size small, results in Stop-and-Go as an attractive solution for providing guaranteed services to real-time traffic.

Stop-and-Go is simple to implement, does not cause congestion, and, has low communication overhead. However, the strict admission control does not allow bursty traffic from utilizing unused resources and bandwidth. Also Stop-and-Go has been

customized for long distance packet networks and needs to be modified for multiprocessor interconnection networks.

2.3.3 Rotating Combined Queuing

Rotating Combined Queuing (RCQ) [13] is meant to be a cost-effective queuing and scheduling algorithm, which can efficiently support a range of service guarantees in multicomputer networks. By allowing bursty traffic to utilize unused network resources efficiently, RCQ can provide competitive performance to best-effort data communications as well. Such cost-effective service guarantees not only provide substantial benefits to the overall system performance, but can also further expand the domain of multicomputer applications to encompass distributed multimedia applications requiring isochronous communications.

RCQ provides a framework to support integrated services efficiently. Unlike other approaches combining separate virtual networks designed for each traffic type, RCQ tightly integrates support for real-time and best-effort traffic in a single network system. RCQ is an appropriate network model supporting a wide range of traffic types including VBR traffic, which requires both guaranteed services and efficiency.

Multiple queues at the NIC's output links are used in order to avoid head of line blocking. Input links have no queues. The RCQ algorithm for queuing and scheduling packets is simple and does not require a complicated NIC.

With a simple frame-based scheduling, however, RCQ also has some limitations in its resource control granularity. The primary drawback of frame-based approaches is that the worst-case packet delay is strongly coupled to the frame size. So, it cannot efficiently support communications with both low delay bounds and low bandwidth requirements.

2.4 Remaining Challenges

The main challenge is to design a multiprocessor server containing an efficient, flexible, fast network and communications mechanism, which caters exclusively to the Video-on-Demand market. Although a lot of research has gone into storage of the videos and the overall system architecture, not much has been done to alleviate all the problems faced by the interconnection network in a real-time Video-on-Demand server.

2.5 Overview of Our Solution

Our work attempts to specifically address the following problem areas of a Video-on-Demand servers' interconnection network –

- Large buffer and storage requirements
- High network bandwidth
- Constant stream rate
- Versatile (fast, flexible and simple) NIC design

We use the following solutions in our proposal for a parallel Video-on-Demand server:

- A flat system architecture with time striping of video blocks across all nodes

- A simple 2-dimensional mesh network topology where each node consists of a processor, a switch and memory (or disk space) for storage of video blocks
- Wormhole routing as the packet switching mechanism
- Deterministic routing algorithms to determine a packet's path through the network
- Buffer queues and virtual channels at the switch's input channels for flow control
- A unique queuing, arbitration and scheduling technique at the switch.

We propose an algorithm called Frame-based Arbitration and Scheduling Technique (FAST). FAST is loosely based on Stop-and-Go scheduling, but does away with the strict admission control policy by allowing burstiness. Static and dynamic virtual channels have been used to manage the congestion caused by allowing burstiness.

FAST is different from Stop-and-Go in that it manages burstiness. The advantage to allowing burstiness is the efficient use of critical unused network resources like buffers and bandwidth. Real-time connections that are close to missing their deadlines can be bursty and send more packets than expected.

FAST is a queuing and scheduling strategy for multiprocessor interconnection networks in Video-on-Demand servers. FAST has been customized to handle transmission of real-time video traffic in multiprocessor interconnection networks. It uses the concept of frames, not for admission control but for congestion management. It can be extended to implement priorities by using multiple frame sizes and prioritized virtual channels.

The FAST algorithm and its merits are explained in detail in Chapter 4.

Summary

In this chapter, we described the architecture of Video-on-Demand servers. We concentrated on describing the problems that the networks of such systems face in order to queue and switch video data. We examined existing research in this area and finally provided an overview of our comprehensive solution to designing an efficient network for a Video-on-Demand server.

Chapter 3

System Model and Architecture

In this chapter we describe the system model used in this thesis. We also describe the simulator used to derive the experimental results.

3.1 System Description

We use the 2-dimensional mesh network topology to organize and interconnect the nodes in the multiprocessor Video-on-Demand server. The Video-on-Demand server has a flat architecture where each node acts as both a storage and a delivery node. Video data is time striped and stored across multiple nodes in the server. The proxy function that is responsible for reordering received video data blocks, is implemented at each node.

NICs or switches forward data from one node to another. Our NIC model assumes only input queuing. Each input physical channel is divided into 5 Static Virtual Channels (SVC) with statically allocated buffers. Each SVC stores packets or flits headed to one of the 5 output channels. The NIC also has a common pool of buffers that are used to construct Dynamic Virtual Channels (DVC) if required.

3.1.1 Network Topology

Multiprocessor network topologies can be classified as either direct networks or indirect networks. A direct network is one in which every node in the network is a processor and is connected to its neighbors via direct links. Popular direct network topologies are the 'n'-dimensional meshes, torus [31] and k-ary 'n'-cubes [37], where 'n' refers to the number of dimensions in the network and 'k' refers to the number of nodes in each dimension. We assume a 2-dimensional mesh model for our work.

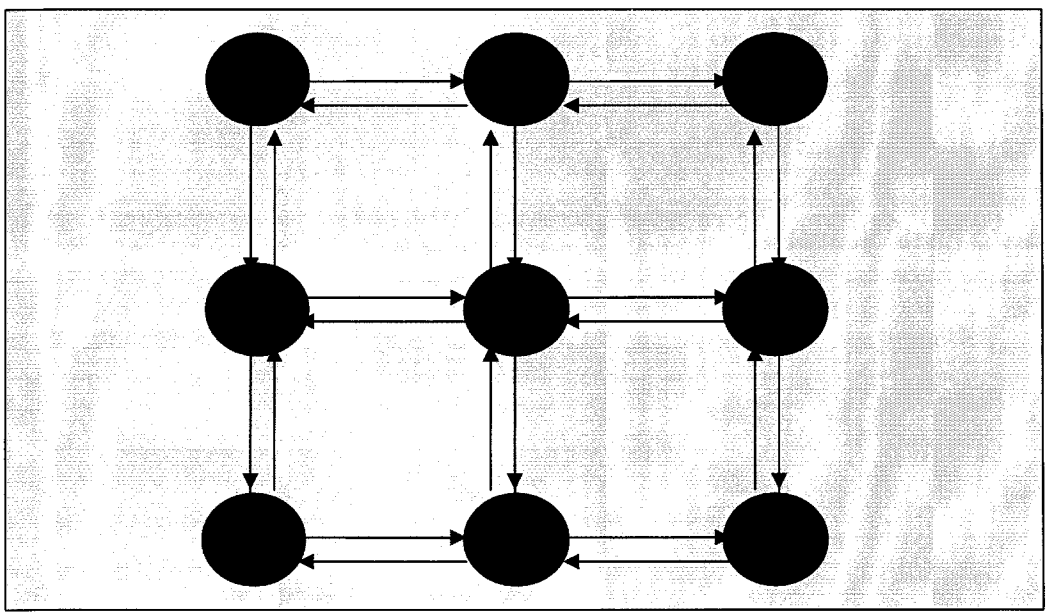


Figure 3.1: 2-Dimensional mesh network

The 2-dimensional mesh network is the most common direct network topology in use currently. This is because of its low degree i.e. 2, which allows simple and efficient, layout and construction with readily available components. Figure 3.1 depicts a simple 2-dimensional 3x3 mesh network with 9 nodes and bi-directional parallel links between the nodes.

3.1.2 Node

A node in a 2-d mesh network is made up of a Processor-Memory (PM) module containing a processor, local memory and a bus connecting them, and a switch that connects the node to the network, also referred to as Network Interface Controller (NIC). As discussed in Chapter 2, Figure 2.6 depicts a typical node in a 2-d multiprocessor network.

The NIC connects the processing node to the network and manages the links to neighboring nodes. It consists of logic to process communication messages and also a set of buffers to hold these messages. Typically the NIC for the 2-d mesh network is a 5x5 crossbar with 5 input links and 5 output links. Four input/output links connect the node to each of its 4 direct neighbors, while the 5^{th} input/output link connects the NIC to the local node. The input links have FIFO queues to store packets/flits that have been blocked from moving ahead in the network.

The NIC performs wormhole routing, which is a special form of cut-through switching. It examines the header of each incoming packet to decide which output channel it must be routed to. If the output channel is free, the packet is forwarded to it; else the packet is buffered in the input queue until the output channel becomes free. The NIC also arbitrates between packets competing for the same output channel. The arbitration policy could be Round-Robin, priority-based or in our case a custom implementation called Frame-based Arbitration and Scheduling Technique (FAST).

The processor handles computation while the switch handles all communication-related tasks so that communication and computation can be handled concurrently at the node.

Communication tasks performed by the NIC include –

- Sending messages in the direction of the destination node (switching and routing),
- Preventing buffer overflow (flow control),
- Removing a packet destined to the local node from the network and sending to the local Processor-Memory module,
- Injecting packets from the local node into the network (switching).

Computation tasks performed by the Processor-Memory module include –

- Processing instructions from the client/user,
- Buffering and forwarding video clips to clients,
- Sending request/response messages to NIC,
- Deciphering the request/response messages received from NIC,
- Aggregating and splitting messages into manageable-sized packets before injecting into the network,
- Sending retrieve request to local memory over the bus,
- Receiving data from local memory over the bus and forming response packets.

3.1.3 Link

A link or channel in a 2-d mesh network connects a pair of nodes and at a physical level consists of a set of wires used to transfer bit-parallel data. The links as shown in Figure

3.1 and as used in our study are bi-directional. The number of wires in a link that transfer data in a direction is defined as the link or channel width.

The NIC performs the function of connecting together the links between neighboring nodes.

3.1.4 Communication Mechanisms

Switching, routing and flow control are important communications concepts that ensure a smooth flow of data through the network.

Switching refers to the process by which a NIC directs packets from its input links to its output links. In doing so, the NICs allocate resources such as buffers and links to the packet as it traverses the network.

Routing refers to the process by which a packet is transmitted from its source node to its destination node. The path taken by the packet to reach from its source to its destination is referred to as the 'route'. The nodes along the packet's route are called hops.

Flow Control mechanisms are usually implemented in hardware, software or both to maintain a continuous flow of packets through the network, thus preventing congestion and deadlock situations. It is the process of adjusting the flow of data from one node to another so that the receiving node can handle the incoming data.

The following sections describe how switching, routing and flow control are implemented in our proposed Video-on-Demand server.

3.1.5 Message

In order to understand switching, routing and flow control mechanisms, we must first understand how movies are stored on the server and how they will be transmitted to the client.

Since an entire movie is too large an entity (from a few GB to several tens of GB) to be stored fully on a node, it is further divided into more manageable blocks and these blocks are striped across multiple nodes. On an individual storage node, the block itself is divided into packets (200 - 400 Bytes) such that each packet translates into an actual movie frame. Each frame of the movie would store information that the client can use to decipher whether the frame is the block's starting frame, one of the middle or body frames, or whether it is the ending frame of the block.

A message in our case is a packet that could be a request message or a response message. Any node in our system could be a requesting node. A requesting node is a delivery node that receives a user's request for a video. When the user asks for a particular video to be played, the requesting node would first determine where the video's blocks are stored and would then send a request message to each node that stores the blocks of the desired video. Upon receiving a request for a video block, the responding node would retrieve

each frame of the block from its local store and would send the response message back to the requesting node.

```
packet structure {
        pknum;          /* Packet number */
        type;           /* Packet's Type – Request or response */
        pkt_next;       /* Packet number of the next queued packet */
        flits;          /* Number of Flits in the packet */
        start_flit;     /* Starting flit's identification */

        connection_no;  /* The connection number assigned by NIC */
        video_id;       /* If response packet, then the video that this packet belongs to */
        block_no;       /* If response packet, then the block number that this packet belongs to */
        block_size;     /* If response packet, then the size of the block that the packet belongs to */
        frame_no;       /* If response packet, then the frame number – start, body or end frame */

        source;         /* Source/Responding node of this packet */
        dest;           /* Destination/Requesting node of this packet */
        next_stop;      /* Next node, NIC or memory id */

        data;           /* If response packet, then the actual video frame data */
};
```

Figure 3.2 Packet structure

Figure 3.2 depicts the typical structure of a message or packet in our server and the key pieces of information required to route it successfully through the network.

The following sections explain the handling of the request and response messages when the requesting and responding nodes inject them into the network.

3.1.6 Switching

We use wormhole routing in our model. When a processor sends a packet to the local NIC, the packet gets further divided into flits. The first flit is the header flit and contains routing information while all the other flits are the body flits. The flits get queued in order at the local NIC's input queue and are switched from this NIC to the next hop along the route in order. When they reach the destination, they are reassembled into a packet and sent to the processor. Figure 3.3 depicts the basic structure of a flit.

```
flit Structure {
        flitnum;        /* Flit number */
        pknum;          /* Number of the packet which was divided to form this flit */
        flit_type;      /* Type of flit – header or body */
        next_stop;      /* Next node or NIC to which the flit is headed */
        dest;           /* The final destination node's id. */
        data;           /* The actual video data */
```

Figure 3.3 Flit structure

The flit size in our system is the channel width or the number of bit carrying wires that are used as links. This width is constant throughout the system. Having the flit size equal the channel width makes it simple for the NIC to switch the entire flit from the input link to the output link in a single system clock cycle.

In our model each input channel of the NIC, including the local processor input channel, is associated with 5 Static Virtual Channels (SVC) such that the first 4 SVCs queue packets headed to the corresponding 4 output channels and the 5^{th} SVC contains packets headed to the local Processor-Memory module. This is shown in Figure 3.4. At each clock cycle, the packets in different SVCs (associated with a physical channel) are

multiplexed and switched to different output channels. This ensures that fewer packets are blocked.

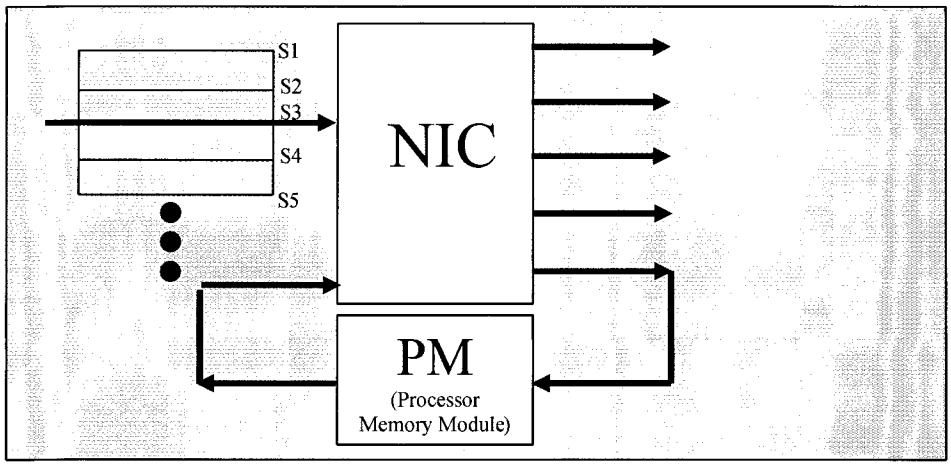


Figure 3.4 Static virtual channels

If the SVCs were not present then all packets incoming to the NIC on a physical channel would get queued to a single queue irrespective of which output channel they are headed to. If the packet at the head of the physical channel's queue is blocked, then all packets queued behind it would be blocked even if they were headed to a different output channel which was available (see Figure 3.5).

In Figure 3.5, the number shown within the packet indicates which output channel it is headed to. Blocked packets are colored black. Consider 2 packets (P1 and P2) that have been switched by the NIC and are in transit to the next nodes and 6 packets (P3, P4, P5, P6, P7 and P8) waiting to be switched.

Figure 3.5.a depicts the normal scenario when there are no SVCs present. In this case packets P3 and P5 headed to output channels 5 and 3 respectively, are blocked since their output channels are busy. Packets P6 and P8 are blocked since packet P3 at the head of

their queue is blocked. Packet P7 is blocked since packet P5 at the head of its queue is blocked. Only packet P4 headed to output channel 2 is ready to be switched. Hence 5 of the 6 packets waiting at the NIC to be switched are blocked.

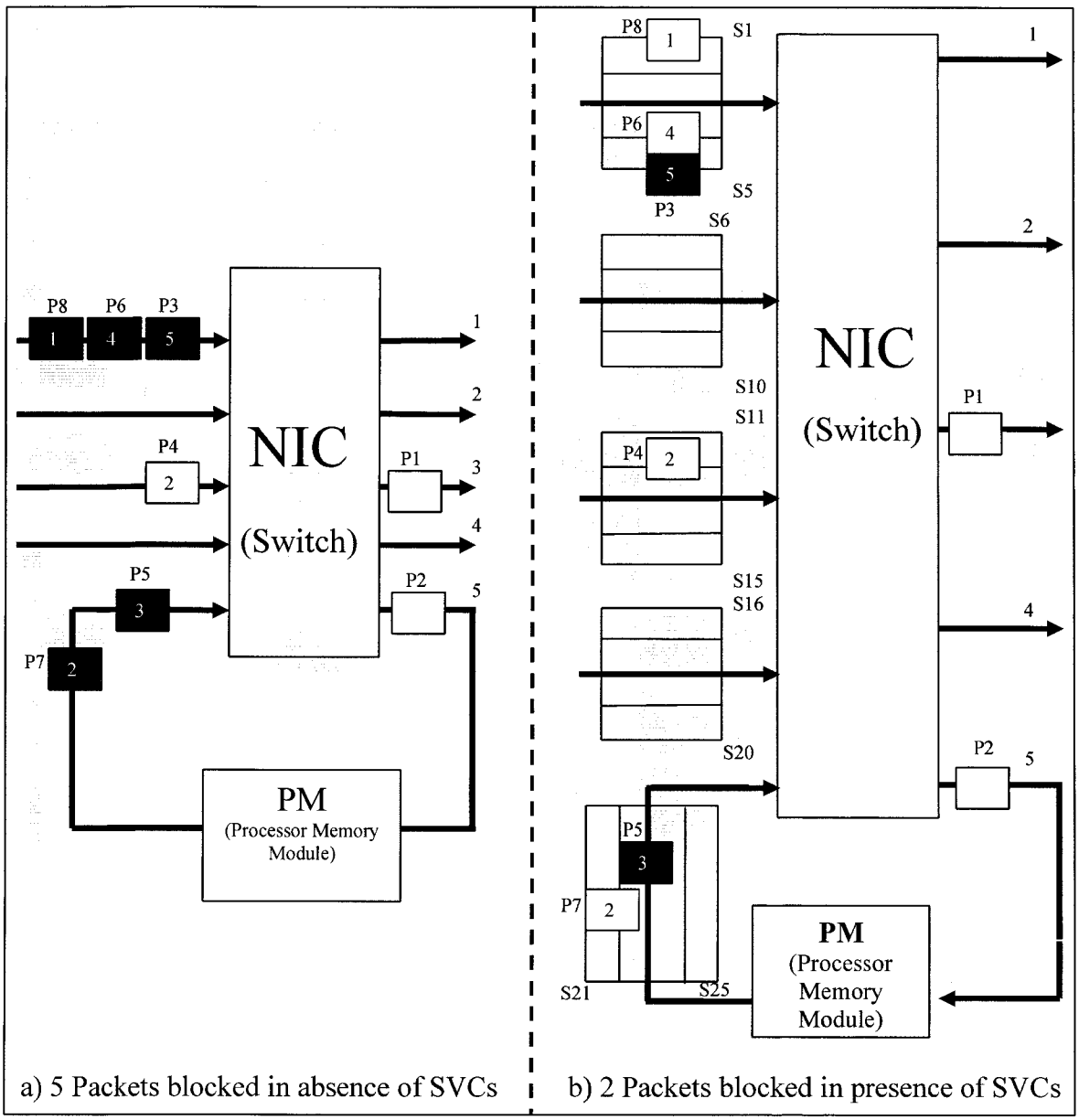


Figure 3.5: Blocked packets and SVCs

Now consider the same scenario but using our proposed NIC design where each input channel is divided into 5 SVCs (see Figure 3.5.b). Incoming packets are queued into SVCs depending on their destination. Packets P3 (in SVC S23) and P5 (in SVC S5) are

still blocked since their output channels 5 and 3 are handling packets P1 and P2 that are in transit. Packet P7 (in SVC S21) headed to output channel 2 is not blocked behind packet P5. At actual switching time packets P4 and P7 may clash since they are both headed to output channel 2. Similarly, Packets P6 and P8 are now not blocked since they are in different queues (SVC S4 and SVC S1 respectively) and are headed to output channels (4 and 1) that are free. Now only 2 of the 6 packets waiting to be switched are blocked.

3.1.7 Flow Control Mechanisms

The following mechanisms are used in our system to achieve flow control –

- Splitting of each input link at the NIC into 5 Static Virtual Channels (SVC),
- Using a queue of buffers at each input SVC in order to queue up packets that cannot be switched immediately to an output channel,
- Making a common pool of buffers available at each NIC to construct Dynamic Virtual Channels (DVC),
- Using the Stop-and-Go concept of delaying packets by one time frame before switching eligible packets to the output links (refer Section 2.3.2).

3.1.8 Routing

We use the deterministic distributed algorithmic routing meaning - the calculation of the packet's route is distributed across all the nodes that the packet crosses in its path from the source node to the destination node.

When a request packet is injected into the network, the NIC at this node will look at the packet's destination node and calculate the next node to forward it to. This NIC then sets up a connection for this request. When the packet arrives at the second NIC, this NIC determines the next stop and stores the connection information in its memory. Each NIC along the request packet's path will follow this process, forwarding the packet to the next node and making note of the connection information.

After the request packet reaches the destination node, the responding node starts injecting response packets into the network. The response packets follow the same path as the request packets. This helps to free the network resources reserved by the request packets and to track connections. A major advantage of using the fixed route from destination to source is the savings in computation time at the NIC especially since response packets are large amounts of video data, with numerous packets being sent to the source. If the NIC were to calculate the route for each such packet, this would be a waste of the NIC's expensive time and resources. Instead, maintaining the connection information at each NIC, immediately tells it where to forward the packet with a simple lookup of the connection information.

3.1.9 Queuing and Scheduling

The following algorithms are used for queuing and scheduling at the NICs:

- Round-Robin
- Stop-and-Go
- Frame-based Arbitration and Scheduling Technique (FAST)

These are explained in more detail in Chapters 4 and 5.

3.2 Simulator

The simulator we use to derive the experimental results uses the SMPL [21] simulation library for generic simulation functions and a set of programs and parameters that are described in [20] which were customized for implementing FAST.

SMPL is a discrete-event simulation language implemented using the C language. SMPL simulates operations using a queuing network simulation model. It works with individual user's main programs in which all the facilities and resources in the system are defined and actions for the events are implemented.

We simulate multiprocessor Video-on-Demand server with a central clock on a cycle-by-cycle basis. Events that occur at clock cycle intervals trigger processing functions, memory lookup and switching functions. Results are collected and analyzed using the batch means method.

3.2.1 Batch Means Method

Batch means method of simulation, also called method of subsamples, divides a long simulation into equal duration parts called batches. A mean of the results collected is computed for each batch and for the overall execution. Standard deviation and confidence intervals for the batches are also computed.

Results for initial batches until the system reaches a steady state could vary highly, hence these initial batches are discarded and these results are discounted. Confidence intervals are calculated for the remaining batches and if this confidence level is high (typically about 95%), it implies a trust in the gathered results.

We use about 30 batches per run and discard the first few batches that have a high variance. Typically, only the first batch is discarded since the batch run time is very large and the system attains steady state during the first batch. Each batch runs for a fixed, very large number of clock cycles (400,000), which happens to be the batch termination criteria. The simulation ends when the 30^{th} batch completes and the system shuts down.

3.2.2 Program-driven Simulation

Our simulator runs the back-end of a Video-on-Demand application on a 2-dimensional mesh multiprocessor system. The application layer is referred to as the front-end where simulated users make requests for playing videos on their client systems. These video requests generate events to trigger the back-end to send request packets over the interconnection network. The request packets upon reaching their destinations trigger response packets to be retrieved from memory and these are sent back to the requesting node. The requesting node forwards the responses to the client. The actual buffering and playing of the video at the client is beyond the scope of this work.

Our simulator is a set of modular functions organized into C language source files. The source has been implemented in a modular way with individual source files for major

components. Some of the major components that have a corresponding source file are listed below –

- Simulation control
- System – Set-up of topology, nodes, links, dimensions etc.
- Inputs
- NIC
- PM – memory and processor
- Request
- Packet – packets and flits
- Workload
- Statistics collection

A text file containing input parameters for simulation is used to drive the program. These inputs are changed for every run of the simulator and results are collected and analyzed for the run.

The program generates output text files storing batch statistics (total packets dropped, average latency and queuing delays, average number of hops), packet statistics (number of hops and total queuing delay per packet), and, channel utilization and queue length statistics. In addition, the debug mechanism can be turned on in order to check the veracity of the run.

3.2.3 Performance Measures

The metrics collected during the experimental run phase are -

- Packet statistics - packets processed (gives an indication of throughput) and packets dropped. *packet loss* (number of packets dropped) is an interesting metric since this figure has to be relatively low so that there is no noticeable jitter when playing the movie at the client.
- Communications statistics - average packet hops, end-to-end latency of movie frames, and packet queuing delay. *End-to-end latency* can be defined as the time elapsed between when a packet is created for transmission at the source node and when the packet is consumed at the destination node. *Queuing delay* in reference to packets can be defined as the time that a packet spends waiting in the input queue of a NIC in order to be scheduled to the output channel.
- Queuing statistics – link utilization, average queue lengths, maximum queue lengths. Link utilization and queue length numbers help to do buffer analysis in order to determine the optimal number of buffers required in virtual channels.

3.2.4 System and Workload Parameters

Unique values for one or more of the following system parameters drive individual experiments. These parameters are defined in an input file and can be changed for the next run without having to recompile the program.

- Number of processors in the 2-dimensional mesh topology.
- Clock cycle time. Memory and network cycle times are assumed to be equal to the processor clock cycle time.

- Maximum requests that can be unfulfilled at a processor. If the maximum outstanding requests value is reached at a processor, then the processor blocks itself and does not accept any new requests until at least one request is fulfilled.
- Scheduling algorithm at NIC - Round-Robin, Stop-and-Go or FAST.
- Time Frame Size 'T', if Stop-and-Go or FAST scheduling is used.
- Queue size of the virtual channels.
- Whether to use dynamic virtual channels.
- Batch parameters - time to run each batch and number of batches to run
- Workload Model – normally, uniform workload is used where there is an equal probability of a processor sending requests to any other processor in the system, including itself.

The workload is defined by the following:

- Request Rate – This is the rate of arrival of requests into the system. It can be calculated based on the input parameter that specifies the interval between requests.
- Request Type – This parameter characterizes the load and is based on the actual requests for videos. We consider 2 types of video requests – balanced or evenly distributed video requests where various nodes request different videos, and hotspot video requests where various nodes all request the same video.

3.2.5 Data Sizes

Movies are typically hundreds of megabytes in size. A 5-minute compressed video is about 2MB in size. This is divided into blocks that are typically 64Kbytes. Blocks are

further divided into frame-sized packets and packets are divided into flits. Simulating the progress of even a 5-minute video through the network leads to hundreds of packets and flits per video request. This massive amount of data cannot be efficiently simulated. Hence, the challenge is to maintain system integrity and simulate as realistic a scenario as possible, while reducing the time to gather experimental results. In order to do this we simulate a system that stores short video clips such as previews and advertisements that play for about 20 to 30 seconds and are about 160 Kbytes in size. These clips are divided into blocks of 16 Kbytes - hence about 10 blocks. The blocks are further divided into frame-sized packets of about 3072 bits. This translates to about 12 flits per frame if the flit and channel size is 256 bits. These simulated data sizes are enough to provide us with a proof of concept for our system.

3.3 A Parallel Video-on-Demand Server

We now present the sample Video-on-Demand server based on the model described above. As shown in Figure 3.6, the system uses a multiprocessor server that is based on the flat architecture model described in Section 2.1.2. The nodes in the server are both storage and delivery nodes, i.e. they store blocks of video data and they're directly connected to a wide area network in order to service clients requesting movies. The flat architecture helps in load sharing and provides surplus network bandwidth.

The blocks of video data are time striped uniformly across all the nodes in the server. Striping helps with load balancing. Each node also contains a proxy module that is responsible for reordering received blocks of video data before forwarding to the client, thus implementing the proxy-at-server architecture. This architecture simplifies the client

systems by doing all the processing of the blocks at server, and presenting them in order to the client. When the client starts receiving the video data, all it has to do is play them.

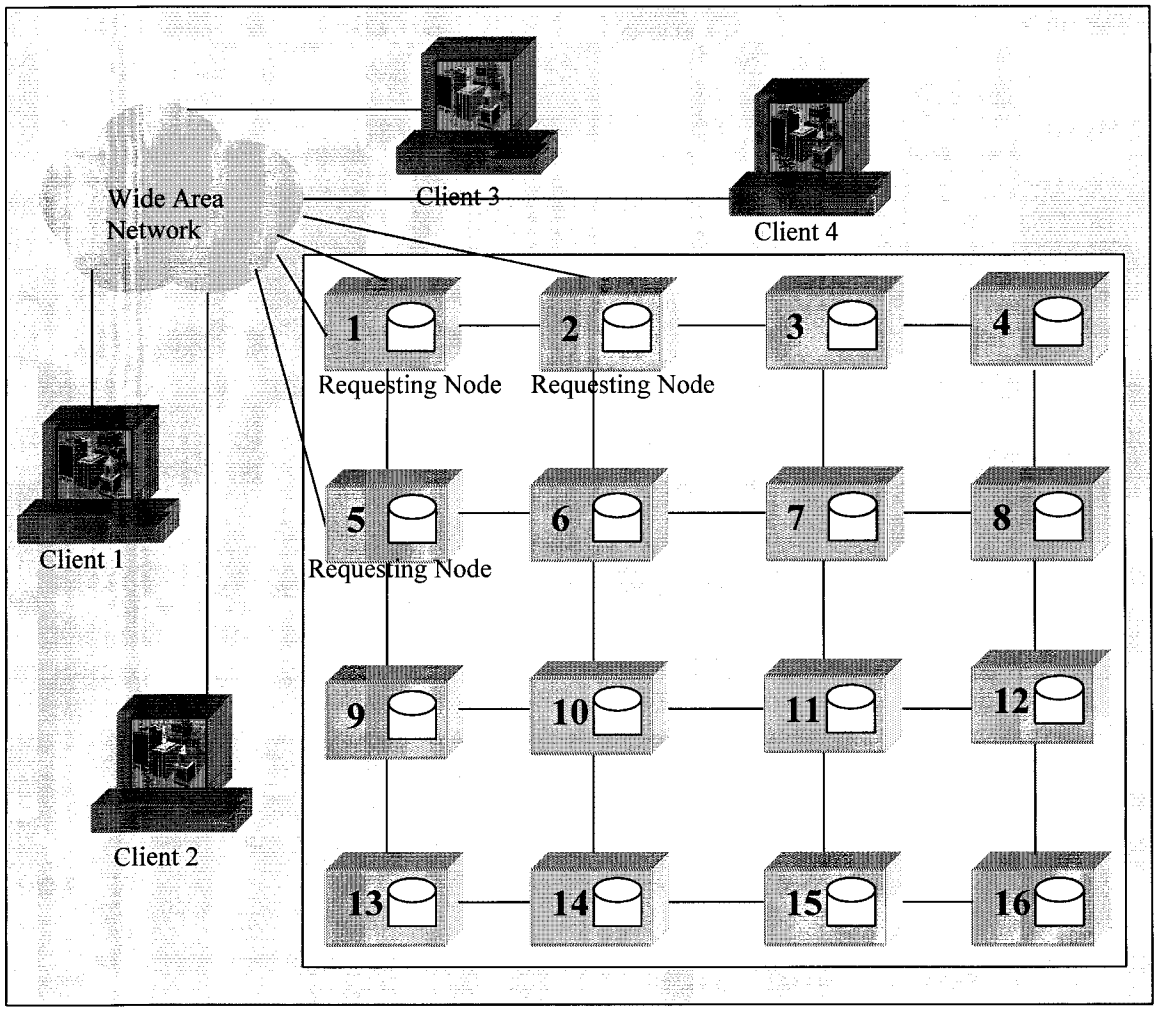


Figure 3.6: A sample Video-on-Demand system

We consider a 2-dimensional mesh topology as described in Section 3.1.2 with 16 nodes organized as a 4x4 mesh. For each workload, we consider a subset of the nodes to be requesting nodes. Requesting nodes receive video requests from clients. They subsequently make requests to all nodes storing blocks of the video to send the blocks. Once a requesting node starts receiving the blocks, the proxy module at this requesting node reorders and stores the blocks in preparation for transmission to the client. When the

first few blocks of the video arrive, the node anticipates receiving the remaining data successfully and starts to send the video data to the client.

The interconnection network in our case uses wormhole routing, thus reducing buffer sizes at NICs and improving packet latency. Packets are routed deterministically using dimension-ordered routing. This is a simple technique in mesh networks that in our case routes packets as far as they will go in the x-direction, before routing them in the y-direction.

The NIC or switch is a 5x5 crossbar that connects the node to up to 4 of its neighbors. The NIC is also connected to the local processor by way of an input and an output channel. It uses the concept of virtual channels where each physical input channel is associated with 5 static virtual channels and a common pool of buffers to construct dynamic virtual channels if required.

Summary

In this chapter, we described in detail the system architecture including the topology and elements of the video server. The communications model and mechanisms used to perform queuing, switching, routing and flow control are explained. We also provided an overview of the simulator used to gather experimental results. Major parameters that drive the simulator and major metrics collected as a result of the simulation are defined.

Chapter 4

Frame-based Arbitration and Scheduling Technique

4.1 Introduction

Most scheduling techniques can be classified as either priority-based scheduling or frame-based scheduling. In priority-based techniques, the switch performs link arbitration based on priorities assigned to packets. The switch assigns more network resources to packets with higher priorities. However, special care needs to be taken to avoid starvation of lower priority packets. Aside from this, most of the priority-based methods do not preserve the first-come-first-served order of the packets. The delays associated to a connection using priority-based methods are difficult to analyze independently, since a particular connection's delay is associated to priorities of other connections. Another disadvantage of such methods is that – as the number of connections increase, the complexity at the switch increases. An example of a priority-based approach is – the Virtual Clock technique [8].

In most frame-based scheduling techniques, the time axis is divided into frames of size 'T'. Connections reserve slots in the frame based on their average rate of transmission.

Usually a strict admission control policy is implemented at the source nodes, so that connections do not overuse reserved bandwidth. All packets that arrive during a particular frame 'Tn' at a switch are considered for scheduling to an output link at the same time. Packets are queued and scheduled in a first-come first-served manner. Since scheduling is done on a frame-by-frame basis, associated delays are easy to calculate analytically. Some of the negative effects of frame-based policies are bandwidth wastage since unused bandwidth can never be used due to the strict admission control policy and the relationship between frame size and associated connection delays. An example of a frame-based approach is the Stop-and-Go algorithm [9, 10].

In this chapter, we present an approach that is an enhanced frame-based technique, except for the fact that it has been tailored to multiprocessor interconnection networks carrying real-time data. This technique called FAST – "Frame-based Arbitration and Scheduling Technique", has been specialized to handle real-time video traffic. The details of the algorithm are presented in the sections to follow.

4.2 Background

FAST is partly based on the Stop-and-Go algorithm, but with the following differences:

1. Stop-and-Go was implemented for long-distance computer networks, but FAST has been customized for short-haul multiprocessor interconnection networks.
2. The network in Stop-and-Go handles best-effort data while FAST is customized to handle real-time data.

3. Unlike Stop-and-Go, FAST does not impose any admission control policies at the source nodes. Hence, even bursty traffic is allowed into the network and extensive burstiness is controlled via a feedback mechanism to the source node.
4. Static and dynamic virtual channels have been utilized at the NICs to avoid deadlocks [6] and head of line blocking. The virtual channels are used for congestion control and can also be used to manage prioritized traffic in future.

4.2.1 Time Frames

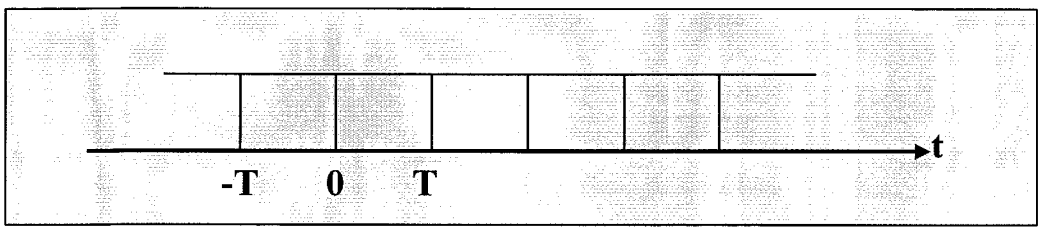


Figure 4.1: Time frames of length 'T'

Both Stop-and-Go and FAST are based on the concept of time frames. In multiprocessor networks, we can assume that there exists a global clock, providing a common reference point in time for all nodes. Starting from a particular point in time, the x-axis is divided into a number of intervals of fixed length 'T', where each interval is called a frame. This is indicated in Figure 4.1.

Packets are allocated slots in the frame and are viewed as traveling with the frame from one end of the link to the other, and then to the switch at the receiving end. When a frame starts off on a link, it is called a departing frame and when it reaches the receiving end, it is called an arriving frame. Arriving and departing frames are synchronized due to the

presence of the global clock. Arriving frames lag behind the corresponding departing frames as shown in Figure 4.2. Using time frames, the total delay of packets is easy to analyze.

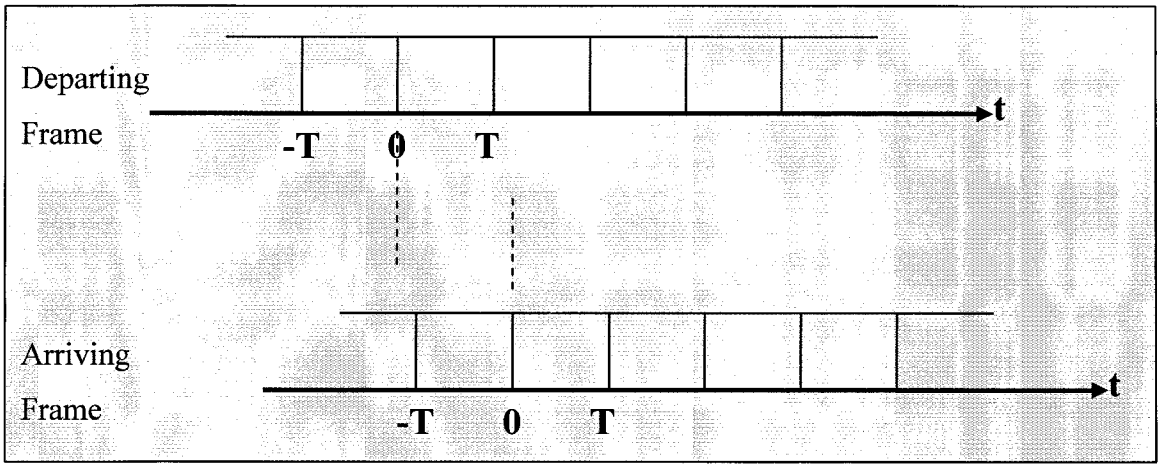


Figure 4.2: Arriving and departing frames

4.2.2 Average Rate/Admission Policy

In Stop-and-Go, a strict admission control policy is imposed on every connection at the source node. Every connection 'K' has an average transmission rate 'r_k' in bits/sec, assigned to it at the source node in the beginning. Supposing that connection 'K' uses a link 'l', then the total length of the packets that can be admitted to the network during a time frame 'T' from link 'l', must be bounded by $r_k * T$.

Maximum frame length in bits = r_k bits/sec * T sec

In other words, during each frame, the total length of the received packets from connection 'K' should be less than $r_k * T$ bits. New packets that would exceed this limit are not admitted until the beginning of the next frame. This policy ensures that the packets of

a connection possess a certain smoothness property when they enter the network from the source nodes.

4.2.3 Smoothness Property

A stream of packets is said to be (r, T) smooth, if during any time frame of size 'T', the total length of the packets admitted into the network does not exceed r*T bits, where 'r' represents the average rate of the traffic and 'T' is the duration of the time frame. This property should be true during any time duration of 'T' seconds and not just those starting at multiples of 'T'.

When a new connection is set-up, it is assigned an average rate 'r_k', and the arrival of its packets to the network should be (r_k, T) smooth. In Stop-and-Go, the imposition of this smoothness property helps to avoid network congestion. However, this initial admission control at the source nodes by itself does not totally prevent packet congestion and must be combined with Stop-and-Go queuing at the switches (see next section), in order to maintain smoothness of the traffic.

4.2.4 Stop-and-Go Queuing

Once the packets from a source node arrive at the input links of intermediate switches or NICs, they are queued and scheduled using the Stop-and-Go queuing scheme. Stop-and-Go queuing alleviates packet clustering and ensures that the smoothness property of the traffic is conserved as the packets move to their destinations in the network. This scheme guarantees that once packets are admitted to the network in a controlled fashion, they

continue to proceed smoothly through the network, thus avoiding packet congestion. Other queuing methods like FIFO (First-In-First-Out) do not preserve the initial smoothness, thus leading to congestion and longer delays.

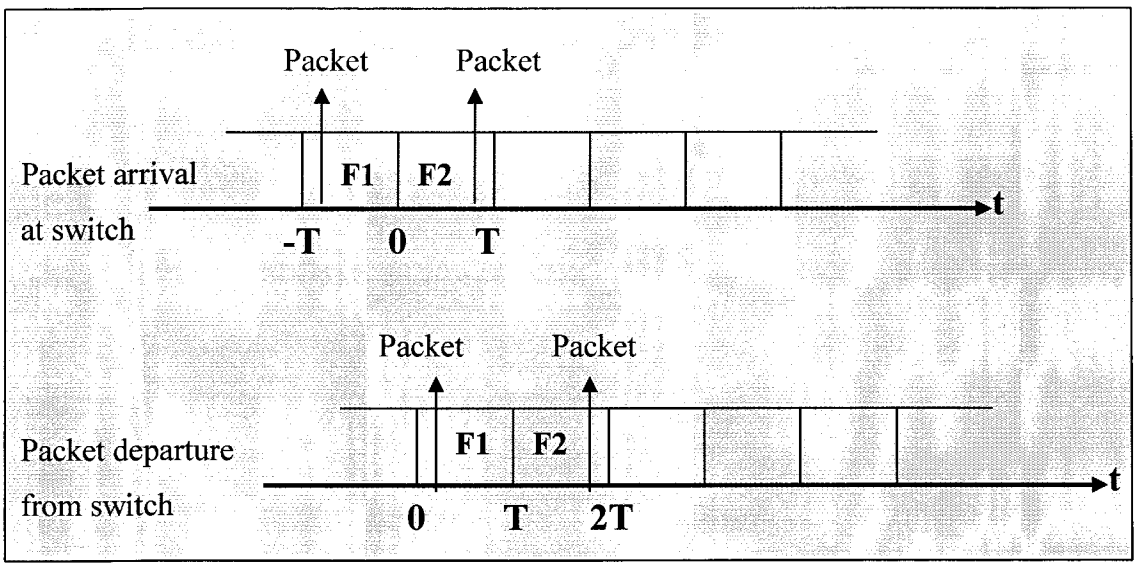


Figure 4.3: Stop-and-Go queuing

The basic idea of Stop-and-Go queuing is to delay the transmission of packets that arrive during a frame F1, until frame F2 starts. Hence all packets arriving during F1 are queued at the NIC's input links and are only transmitted during the next frame F2. Also, only those packets that arrived before or during F1 are transmitted in F2. Any packets arriving at the input links during F2 are delayed by one time frame 'T', even if there is room to transmit them during F2. Therefore, packets that arrive during a frame are only eligible for transmission during the next frame, as shown in Figure 4.3.

The scheme guarantees the following:

- A packet that arrives on an input link during a frame 'F', will be transmitted before the end of the frame after 'F'.
- The packets from each connection will maintain the (r, T) smoothness property during their lifetime.
- The total queuing delay of a connection is bounded by H*T and 2*H*T, where 'H' is the number of hops traveled by the packets of the connection. This constant and reasonably low delay value is what makes Stop-and-Go such an attractive solution for providing guaranteed services to real-time traffic.

FAST removes Stop-and-Go's strict admission restriction and adds the concept of virtual channels. The following sections discuss FAST's design details.

4.3 Definitions

The following definitions are specific to our implementation and will help in better understanding the algorithms.

Requesting Node can be defined as the node that sends out the request for the video to be played.

Responding Node can be defined as a node that stores at least one block of the video being requested. This node sends the video block to the requesting node in the form of packets containing frames of the movie.

NIC or the Network Interface Controller (also referred to as the switch or the router) connects a node to its neighbors and makes decisions about which neighbor to switch a packet to based on a deterministic routing algorithm.

Request set-up packet is a special type of control packet that is sent from the requesting node to all the responding nodes storing blocks of the video being requested. Major pieces of information contained in this packet are the video id being requested, source and destination node addresses.

Response packet is a data packet containing a frame of the video being requested and is sent from a responding node to the requesting node.

4.4 FAST Queuing

4.4.1 The FAST NIC

Queuing and transmission of packets to output links in FAST is different from Stop-and-Go. Figure 4.4 depicts a NIC with network input and output links or channels. When packets arrive at the switch for routing, they get queued in one of the network input link's buffers or queues. Each network input link is associated with further 5 Static Virtual Channels (SVC).

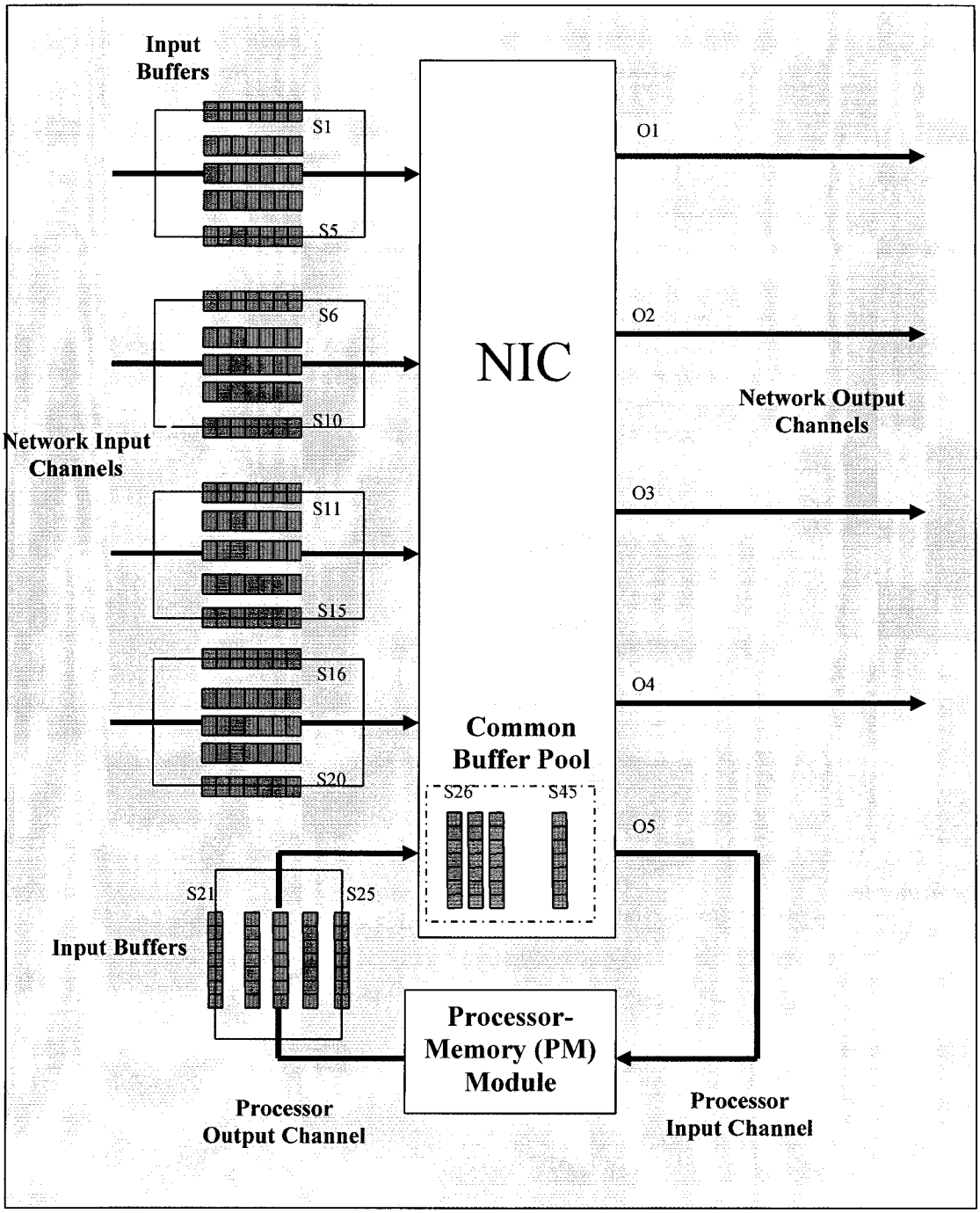


Figure 4.4: The FAST NIC

The switch also has physical input and output links to its local Processor-Memory (PM) Module. Here too the input link is associated with 5 static virtual channels. The SVCs of

each physical link store packets for different output links – O1 to O5. Hence, if we number the SVCs for all 5 input links from 1 to 25, SVCs S1, S6, S11, S16 and S21 store flits for network output link O1; SVCs S2, S7, S12, S17 and S22 store flits for network output link O2; SVCs S3, S8, S13, S18 and S23 store flits for network output link O3, SVCs S4, S9, S14, S19 and S24 store flits for network output link O4, and SVCs S5, S10, S15, S20 and S25 store flits for processor output link O5.

Apart from the input and output links, the switch has a common pool of buffers that are used for creating Dynamic Virtual Channels (DVC) as and when required. These DVCs are labeled starting from S26. Figure 4.4 has 20 DVCs in its pool of buffers. The last DVC is labeled S45.

4.4.2 Virtual Channels in the FAST NIC

Each packet consists of flits. Flits of one packet cannot be interleaved with flits of another packet. Hence there is no gap in the transmission of all the flits of a packet. In the interest of simplicity, in future sections, we will not refer to flits but to the packet as a whole.

Each SVC has it's own buffer queue for queuing the packets that arrive regularly during a time frame 'F' of duration 'T'. All corresponding SVCs of each input link are multiplexed and switched to the output channel in the same clock cycle, as shown in Figure 4.5. If everything is smoothly regulated, these frame-size buffers are sufficient for storing the packets arriving during any frame. However, since FAST does not impose any

admission control, we employ DVCs to store additional packets from a bursty connection.

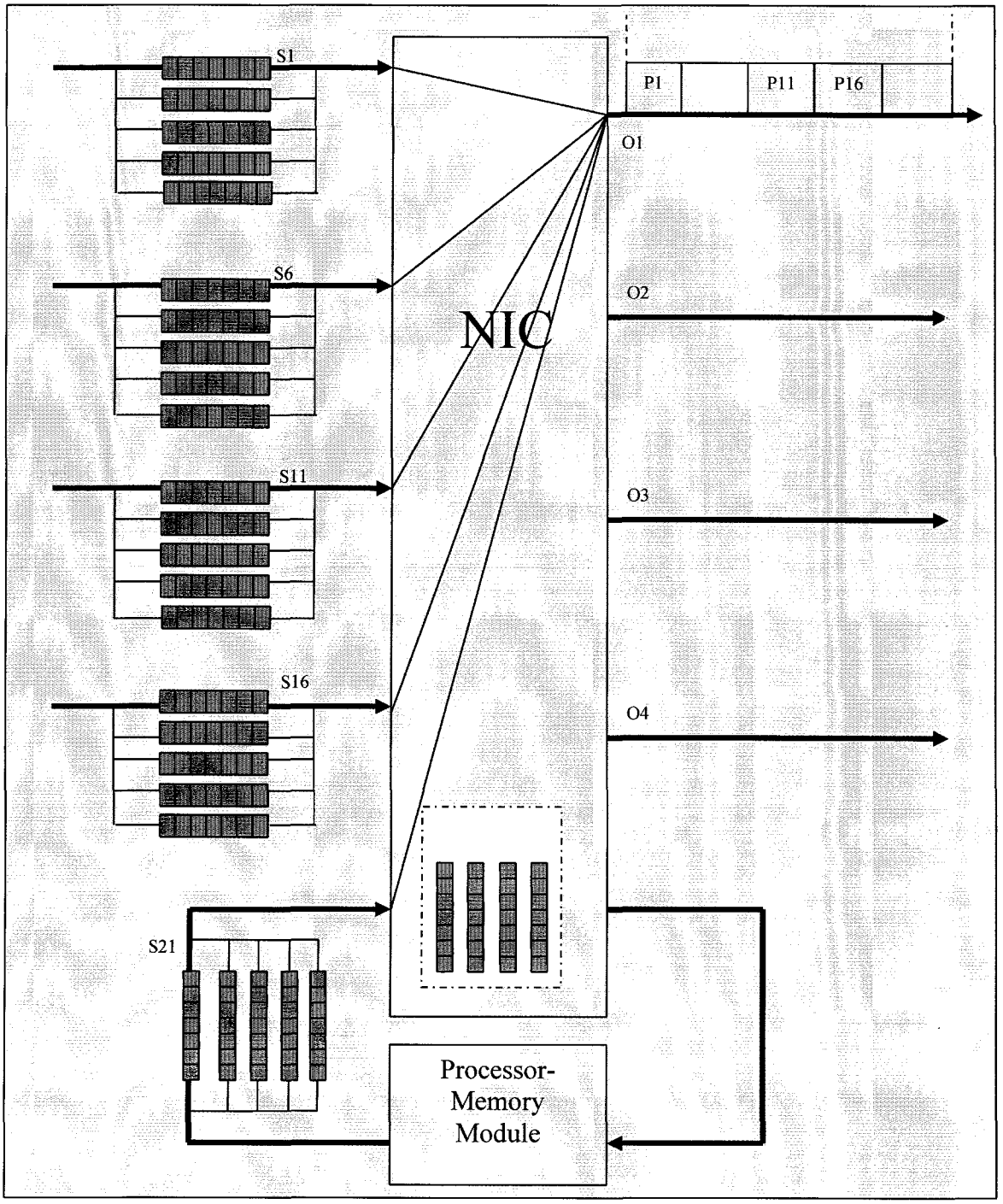


Figure 4.5: Multiplexing and transmission of packet queues

Buffer#	Available	Associated SVC
1	0	S1
2	1	0
3	1	0
4	1	0

Table 1: Common buffer pool table

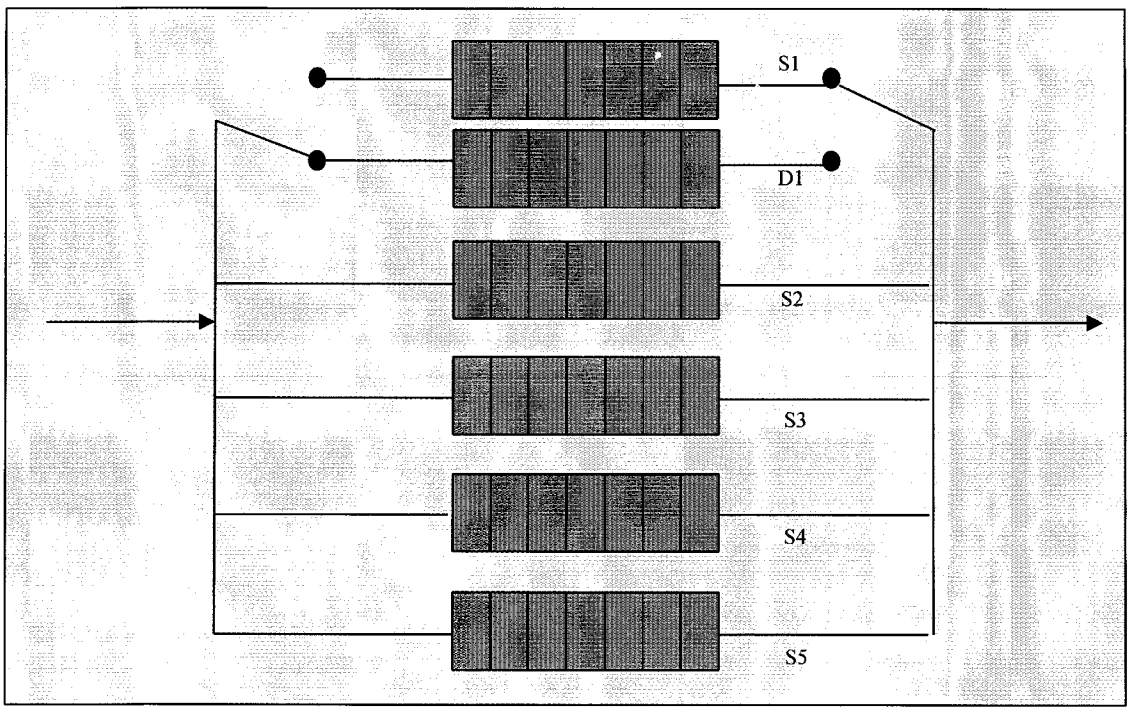


Figure 4.6 Dynamic virtual channels in the FAST NIC

Every FAST NIC has access to a common buffer pool with frame-sized buffers. This buffer pool is managed with the help of the buffer table as shown in Table 1. When a bursty connection fills up it's allocated SVC, for example, SVC 'S1', and is still sending more packets, the buffer table is checked for an available buffer set. If any buffers are free, they are borrowed from the common pool and a virtual channel is created dynamically for storing the bursty packets. This new DVC, say 'D1' is associated to the

same output link as 'S1', which is output link 1. Any additional packets headed to 'S1' will now be stored in 'D1', as shown in Figure 4.6.

4.4.3 Connection Information

When a node requests a video to be played, request set-up packets are sent through the network to the nodes that store the video's blocks. As this request set-up packet traverses the switching nodes or NICs, the switches consult their respective connection tables, as shown in Table 2, and indicate whether they can accept this new connection or not, by setting a bit in the request set-up packet. If the NIC can accept the connection, it makes an entry for the connection in the connection table and updates the *packets_queued* field (total number of packets from a particular connection queued at this NIC) to –1.

Connection #	Average Rate	Source Node	Input Link	Output Link	Packets Queued
K1	150	2	S1	O1	4
K2	100	5	S1	O1	2
K3	175	13	S10	O2	12

Table 2: Connection table

When a request set-up packet reaches a node that stores a block of the video, this node looks at it's own available resources and at all the bits set by the intermediate switching nodes to determine whether the connection can be accepted. If the connection cannot be accepted, the source node sends a NACK (Negative ACKnowledgement) packet back to the requesting node. The NACK being a very small packet traverses the network very quickly and arrives back at the requesting node before any of the response packets. The NACK packet takes the same route as the request set-up packet and as it passes the NICs, it removes the entry for this rejected connection from each NIC's connection table. If a

NACK is received from any node storing a block of the movie, the requesting node will refuse the user's request for playing the video.

However, if the node accepts the connection, the video block is transmitted to the requesting node by dividing the block into more manageable-sized packets. These response packets follow the same route as the request set-up packet. As the NICs start receiving packets from the connection, they update/increment the *packets_queued* value. The last response packet for a connection cleans up the connection information table as it passes through each NIC along the route to the requesting node.

4.4.4 Packet Timestamp

When a packet arrives at a NIC, it is queued into one of the virtual channels. At the same time, the packet is stamped with it's arrival time at the NIC. The current timestamp value at each NIC is stored locally at the NIC and is just a counter value that is incremented with each clock cycle. When the maximum value of the counter is reached, it is reset to zero.

A packet's timestamp helps to determine when it is eligible to be scheduled or switched to the output channel. This relates back to Stop-and-Go's concept of eligible packets. In the next section we will see how the timestamp is used for FAST scheduling.

4.5 FAST Scheduling

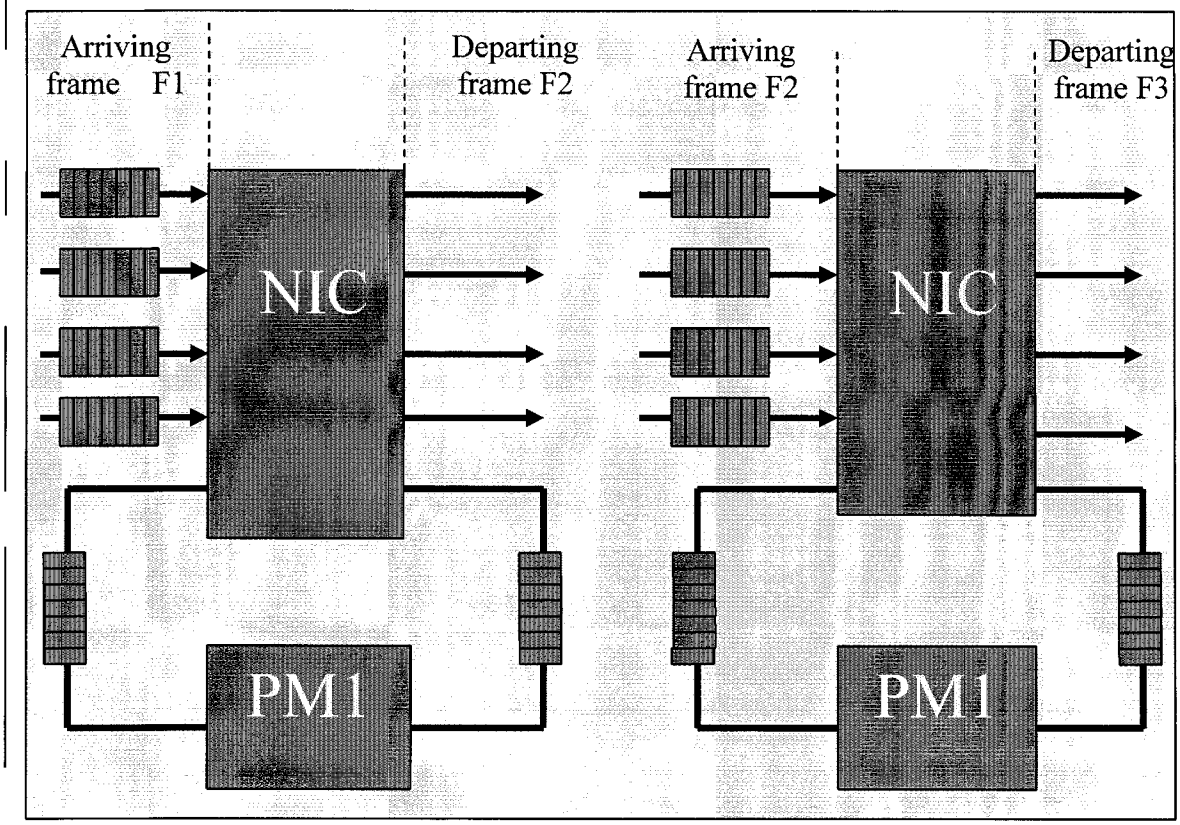


Figure 4.7: Arriving and departing frames

One of the most important concepts of Stop-and-Go and FAST is that the switch always delays the transmission of packets by one time frame. All packets that arrive during a frame F1 are eligible for transmission during the next frame F2. Such packets are called eligible packets. Hence, if a packet arrives during a time frame F1, it is scheduled to the output link only during frame F2, which is the frame immediately following F1. Frame F1 in this case is the arriving frame and if the packet is scheduled during F2, frame F2 is the departing frame, as shown in Figure 4.7.

4.5.1 Packet Servicing and Queuing Delays

The above scenario works fine for as long as all connections transmit packets regularly in such a way that injection of packets into the network by source nodes is limited by the average rate they have specified for the connection. Under such conditions, FAST's delay characteristics are bounded and will be very similar to Stop-and-Go's delay characteristics.

This constant and low queuing delay is excellent for the quality of service guarantees that video traffic requires. However, since FAST allows burstiness, bursty traffic from some connections is allowed into the network. In such cases, when packets arriving at a switch run out of buffer space in their allocated SVC, a DVC is created and packets from all connections coming to the fully queued SVC get queued in the DVC.

In the presence of DVCs, we have to remember that each input SVC can have only one additional DVC. If both the SVC and DVC get filled up before either of them is processed, and more packets are headed to this link, then these packets have nowhere to be buffered and must be dropped.

If an SVC has an associated DVC, there exists an additional input link that needs to be serviced. If an input SVC has an associated DVC, then at scheduling time a decision needs to be made about whether to service the SVC or the DVC. In this case, the timestamps of the first packet in the SVC and the first packet in the DVC are compared.

If both packets are eligible then the packet with the earlier timestamp is switched, else the packet that is eligible is switched. This algorithm is given in Figure 4.8.

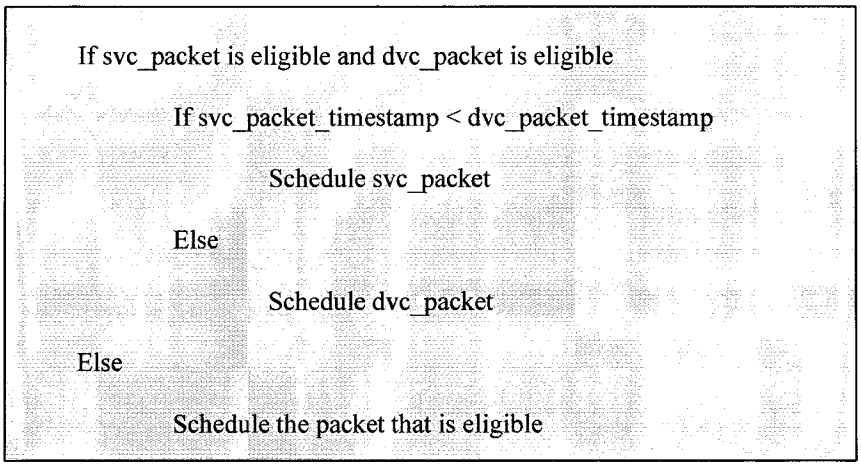


Figure 4.8: Algorithm for priority of servicing of virtual channels when both SVC and DVC are present.

Suppose if during frame F1, connection k1's packets are queued in SVC S6 along with packets from connections k2 and k3. If k1 is bursty and sends more packets than it's average rate, then packets from connections k1, k2 and k3 fill up SVC S6 and now start filling up a newly created DVC – D6. When it is time for scheduling, packets in S6 that arrived during F1 are scheduled before packets in D6 that also arrived during F1, due to their earlier timestamp. So, S6 is emptied earlier than D6 initially. However, future packets queued into S6 that arrive during frame F2 have later timestamp than some of the packets in D6. After DVC D6 is emptied, if SVC S6 has room for packets, the DVC is dropped and the buffers go back to the common pool of buffers.

Now, even though SVC S6 contains packets that arrived during frame F2, these packets may not get serviced during F3 as expected, but may only get transmitted during frame F4 or later, due to the packets in the DVC being processed earlier. Therefore in the

presence of burstiness packets can get delayed and cannot maintain Stop-and-Go's delay characteristics. Chapter 5 provides results for FAST's delay characteristics obtained by means of simulation.

4.5.2 Feedback to Bursty Source Nodes

Sometimes allowing unlimited burstiness for certain connections is inefficient since the other connections that transmit regularly get delayed due to lack of resources. The connection tables in the switches along with an algorithm in the switch for checking which connections are being bursty help to control these greedy connections.

When a DVC gets created, the switch monitors its capacity. If the DVC gets filled beyond a certain threshold, the switch checks it's connection table to determine which connection is being bursty. It then sends a feedback signal to the source node of the bursty connection in order to get the node to reduce its transmission rate. The feedback signal is implemented either as a small control packet transmitted to the source node or it is implemented in the hardware as a single wire that is raised when feedback is to be given.

The feedback signal is necessary in order to avoid packet loss. Since there is only one DVC for every SVC, if packets fill up the DVC as well as the SVC, any additional packets have to be dropped. This could include packets from non-bursty connections headed to the same virtual channel. To avoid this packet loss and subsequently a jitter when the video is played at the destination, the feedback signal is required.

The threshold value for the DVC will initially be set at 50% of the DVC's capacity and should be changed to an optimal value after experimentation.

4.6 The FAST Implementation

The FAST algorithm can be divided into 3 parts and viewed from the point of view of – the requesting node, the responding node and the NIC itself. This section outlines the pseudo-code for the three portions of the algorithm.

4.6.1 Data Structures for Implementing FAST

Tables 3 and 4 provide details for 2 of the data structures (open requests and movie information) required for implementing FAST. These two tables in conjunction with the packet data structure shown in Figure 3.2, flit data structure shown in Figure 3.3, common buffer pool information shown in Table 1 and connection information shown in Table 2, store all the major pieces of information required for implementing FAST.

Movie Id	Block #	Destination
1	1	8
	2	15
2	1	1
3	1	4
	2	7
	3	2

Table 3: Open requests table

Each PM keeps a record of all its outstanding requests in the 'open_requests' table (Table 3). This is just a simple structure that contains the movie id that the client has requested, the block # and the responding node # where this block is stored.

The movie information table (Table 4) is a global data structure available to all PMs. It has a record of all the videos and blocks that are stored on the system. The fields in this table are the movie id, the number of blocks that the video has been divided into, and information about the block. For each block, the table provides information on its number, how many video frames are contained in it and which node it is stored on.

When a client request comes into a requesting node, this node refers to the movie information table to determine where the blocks of the video are stored. It then sends a request to each node that stores a block of the video.

Video Id	# of Blocks	Block#	# of Frames	Destination
1	3	1	40	4
		2	50	7
		3	50	2
3	2	1	42	8
		2	45	15

Table 4: Video information table

4.6.2 Requesting Node Algorithm

The requesting node is the one that makes the request for the video to be played. When it makes requests for video blocks, these video blocks stored at different nodes make their way through the network to this node. Hence the destination for the blocks is the requesting node. Requesting node algorithm is shown in Figure 4.9.

Requesting a video:

- Randomly select the video to be played
- From 'movie_info' table, retrieve '# of blocks', 'block #', 'destination/responding node'
- Store video information into 'open_requests' table
- For each block of the movie:
  - Create a request packet with 'video id', 'responding node' information, structure for storing whether intermediate switches will accept connection or not, and request type 'RT_REQ_SETUP'
  - Send packet to 'responding node'.
  - In case 'responding node' is the local node, just retrieve block from memory.
  - Else,
    - Split request packet into header and flits
    - Send request on its way to 'responding node' by queuing request packet and flits into one of the input queues of the NIC.

Processing response packets:

- When response packet is received, check type of response
- If response packet type is a NACK:
  - Display a message saying - "Please try again"
  - Decrement number of outstanding requests
  - Remove entry for this request from 'open_requests' table
  - Close the open request and clean up
  - Schedule the next request
- Else if response packet is the start frame of the movie:
  - Prepare to assemble the blocks of the movie from the frames
  - If proxy-at-server module is implemented, reorder received blocks
  - Forward blocks to the client for playing the movie
- Else if response packet is one of the body or middle frames of the movie:
  - Ensure that frames are received in order
  - Buffer the frames so that the blocks can be reassembled
- Else if response packet is the end frame of the movie:
  - Decrement number of outstanding requests
  - Remove entry for request from 'open_requests' table
  - Close the open request and clean up

Figure 4.9: Requesting node algorithm

4.6.3 Responding Node Algorithm

Handle request:

If the packet just received by PM is a request packet, i.e. packet's type is RT_REQ_SETUP

- Check that all NICs along the route have accepted the connection. Else send NACK packet with type of 'PK_NACK'
- Queue the request to the appropriate memory bank that stores the response block requested
- Release the memory and assign it to the waiting packet
- Calculate how many memory cycles are required to retrieve the block
- Lock the processor-memory bus for the required cycles during which the block is fetched from memory
- From 'movie_info' table, determine how many frames the movie block is divided into. For each frame of movie, invoke function 'Send response'

Send_response:

- Split the video frame into packet header and flits, if headed to next hop
- Set the destination for this packet to be the requesting node
- Set response type to be 'RT_RESPONSE'
- If this is the first frame of the block, set frame type to be 'START_FR'; else if this is last frame, set frame type to be 'END_FR'; else set frame type to be 'BODY_FR'
- Set 'connection #' to be the same as request packet's 'connection #'. This is so response packet can follow same route as request by looking up 'connection-info' table to check its next hop
- If response is destined locally, notify local processor
- If response is destined for remote PM,
    - Prepare packet for network access
    - Set packet's next stop as the next hop NIC along the route to its destination
    - If there is space in NIC's input queue, queue packet and flits into NIC's input queue.

Figure 4.10: Responding node algorithm

The responding node is the one that stores the video blocks and forwards them upon request. When this node receives a request, it checks to see whether all the intermediate switching nodes have accepted the connection. If they have then, it retrieves the video blocks for the movie from its storage, forms packets for it and then injects the packets into the network. The details for the algorithm are depicted in Figure 4.10.

4.6.4 NIC Algorithm

At each network cycle, if time is a multiple of frame size, increment NIC's 'frame #'

(1) Queue transit packets to incoming queues:

- For each input channel coming from other NICs –
  - Examine packet(s) (max 5) in input channel - if any
  - If packet's type is RT_RESPONSE, then get its next hop from 'connection_info' table, so that response packets can follow same route as request packets
  - If packet's type is RT_REQ_SETUP, call routing function to determine the next hop
  - FAST QUEUING: Queue the packet in one of the 5 SVCs based on which output channel the packet should be scheduled to
  - Determine virtual channel for queuing packet. Normally, this is the SVC, but if SVC queue is full, call function to allocate DVC from central pool of buffers
  - If chosen virtual channel is full, drop the packet and all its flits
  - Else
    - Queue packet into chosen virtual channel
    - Timestamp packet with NIC's 'frame #' (used later for eligibility checking).
    - If packet header, and flits are following, lock queue and reserve it for this packet
    - Else, if last flit of packet, unlock queue

(2) Process packets in incoming queues (SVCs):

- For each SVC (0 to 24)
  - If there is a packet in SVC, find corresponding DVC - if any. Determine whether to schedule SVC or DVC
  - Check packet's timestamp to determine if it is eligible to be scheduled (packet should have arrived at NIC in previous time frame, or before that)
  - If output channel is free, i.e. not locked by a different packet –
    - If this is a request packet, update 'connection_info' table with next hop info
    - Else, if this is the last response packet for a particular request, delete entry for this connection from NIC's 'connection_info' table
    - Send packet from chosen virtual channel (SVC or DVC) to output channel
    - If no more packets in input DVC queue & DVC is in use, release DVC

(3) Process packets in local output channel (The 4 other output channels become input channels for neighboring nodes. These channels contain transit packets that are processed in part (1)):

- If there is a packet in the output channel headed to the local PM
  - Merge flits and header into a single packet
  - Notify PM about the arrival of a packet

Figure 4.11: NIC algorithm

The intermediate NICs in the path between the responding and requesting nodes perform all the networking and communication functions. These switches have to process request set-up packets, create and update connection information in the connection table, queue packets in the appropriate SVCs, create and destroy DVCs as and when required, schedule packets to output links, coordinate time frames, and, decide when to provide feedback to bursty source nodes when feedback mechanism is implemented. Figure 4.11 details the actions taken by the NIC for simulating FAST.

Summary

This chapter explained the FAST algorithm in detail. In summary FAST forms an efficient framework for forwarding and controlling video traffic through multiprocessor networks, with minimum delays and low to non-existent packet miss ratios. Virtual channels, queuing and scheduling combined with feedback mechanism to bursty nodes are some of the important concepts used by FAST to limit packet delays to very low rates.

Chapter 5

Performance Analysis

We analyze the simulation results for a multiprocessor Video-on-Demand server that has a 2-dimensional mesh topology. As a part of this performance study, we compare the performance of our proposed queuing and scheduling technique, namely FAST to two other well-known scheduling techniques: Round-Robin and Stop-and-Go.

We will clearly establish the following results to prove that the proposed FAST algorithm is better at handling real-time multimedia traffic compared to Round-Robin and Stop-and-Go:

- FAST is more capable of handling bursty hotspot conditions (conditions under which a popular video is requested over and over again), than Round-Robin and Stop-and-Go
- FAST provides lower end-to-end latency and higher throughput than Round-Robin and Stop-and-Go
- FAST has 300% less packet queuing delays compared to Round-Robin
- FAST has at least 200% lower end-to-end latency than Round-Robin and 500% lower latency than Stop-and-Go

We will compare 3 different time frame sizes (T = 1, 5 and 10 clock cycles) within FAST in order to see if varying the time frame size has any effect on performance and to provide an engineering guideline on choosing frame-size. We will observe that (i) under hotspot conditions, when there is congestion at some nodes and (ii) at higher loads, larger time frame sizes (T = 5 and 10 clock cycles) perform better than small time frame size (T = 1 clock cycle). At lower loads where congestion is minimal, performance is independent of the frame size.

We will show that when queue size is reduced FAST performs consistently better than Round-Robin (in terms of number of packets dropped). When we introduce Dynamic Virtual Channels, the number of packets dropped in FAST is comparable to Stop-and-Go, in addition to FAST having a much higher throughput than Stop-and-Go.

We will perform buffer analysis based on the average and maximum SVC queue sizes, for the worst-case load. We will show that there is a good probability of buffer usage being higher than average. Hence, using the average value as opposed to the maximum value for SVC queue size will lead to significant under-engineering of the system, thus causing packet loss.

All the simulation results have a confidence interval of +/–1% at a 95% confidence level, except for Round-Robin where the confidence interval increases to +/-5%.

5.1 Algorithms

5.1.1 Round-Robin

Round-Robin is a legacy queuing and scheduling technique that queues packets in a first-come-first-served basis into one of several queues, and then schedules packets at the heads of queues one after the other in a circular manner. This is a simple technique requiring minimal processing overhead at the NICs. Round-Robin however has no congestion management and its performance will start to deteriorate at medium to high loads.

5.1.2 Stop-and-Go

Stop-and-Go is a frame-based technique used for congestion management in integrated services networks. Stop-and-Go employs a strict admission policy that does not allow bursty traffic from entering the network, resulting in bounded queuing delays and loss-free transmission. The concept of time frame is central to this technique where time frame is defined as a unit of time in which connections reserve time slots based on their transmission rates. The stringent admission control allows only as many packets per connection as time slots that have been previously reserved. This admission requirement tends to waste precious network resources in cases where some connections do not fully use their reserved bandwidth.

Stop-and-Go queuing and scheduling at the NIC uses time frame to determine the eligibility of a packet in an input queue for transmission to the output channel. Scheduling of packets from input to output channels is delayed by one time frame.

Together the admission policy and Stop-and-Go queuing maintain a certain smoothness to the flow of traffic thus avoiding congestion and providing good service guarantees in terms of queuing delay and buffer sizes.

We will show that Stop-and-Go's strict admission control leads to long end-to-end latencies for video frames at any load condition. Upon receiving a request, the processor accesses its memory to retrieve video blocks. Even though packets are created quickly, these packets wait a long time to gain access to the network. Thus bursty traffic is disallowed even though resources are available for handling this traffic. Since a lot of real-time traffic is bursty, this algorithm in its basic form is not suitable for Video-on-Demand servers.

5.1.3 Frame-based Arbitration and Scheduling Technique (FAST)

FAST is an algorithm designed specifically for multiprocessor NICs. Similar to Stop-and-Go, FAST is based on the concept of time frames and delaying the scheduling of packets by one time frame. However, the advantage of FAST is that it allows bursty traffic into the network, thus mimicking a real-life multimedia server. FAST handles congestion at NICs by using virtual channels. The SVCs prevent head-of-line blocking and at the same time increase buffer capacity at NICs. In addition, a pool of DVCs is allocated on an as-needed basis. By adding to the SVC's capacity, DVCs effectively handle bursty traffic.

Three time frame sizes (1, 5, and 10) are considered for simulation. When time frame size is 1 network clock cycle (T1), packets that arrive at a NIC's input channels are eligible

for transmission to the output almost immediately. Hence, no additional delay is introduced in order to smooth out the traffic and avoid congestion. When time frame size is 10 clock cycles (T10), packets arriving at a NIC's input channels are not eligible for transmission to the output channels until the beginning of the next time frame, which is at least 1 clock cycles and at most 10 clock cycles away.

We see that for all 3 time frame sizes, FAST performs consistently well across all load and video request conditions. Its queuing delay although higher than Stop-and-Go is still low and superior to Round-Robin. End-to-end latencies are superior to both Round-Robin and Stop-and-Go, proving that FAST can effectively handle bursty traffic. FAST's consistency makes it ideal for use in real-time, multimedia systems. Packet loss analysis for FAST shows that even though number of queues required is more, queue sizes are small.

We also observed that adding additional delay to packets by increasing time frame size is advantageous in some cases.

5.1.4 Implementation

Our system implements the FAST algorithm as outlined in chapter 4. In order to implement Round-Robin, Stop-and-Go and FAST using the same system and with minimum changes, an algorithm parameter was added to the input file. This parameter governed all the changes for the system in order to simulate Stop-and-Go, Round-Robin or FAST. Another parameter for time frame size 'T' was also used.

Some of the differences between the implementation of the algorithms is given below:

- Round-Robin and Stop-and-Go do not use virtual channels both static and dynamic. Hence, the number of NIC input queues were made a variable and its value was set to either 5 (for Round-Robin and Stop-and-Go implementations) or 25 (for FAST's SVC implementation). A Boolean variable governed the use of DVCs based on the algorithm being executed.
- An admission control policy was implemented for Stop-and-Go.
- FAST and Stop-and-Go use the concept of time frames for congestion management. Packets waiting in input queues of NICs are delayed from being scheduled to the output channels by one time frame. Round-Robin does not check packets for eligibility but straightaway schedules them to output if resources are available. Hence, eligibility checking was implemented if either Stop-and-Go or FAST was being executed.

5.2 Load Characterization

The analysis performed here is based on uniform arrival of requests where the arrival rate is varied to simulate low, medium and high loads on the system. Although there is no admission control at the ingress to the network in fast, there is a request control mechanism that disallows the processor from accepting new requests if more than a certain number of requests are pending for that node. This is called blocking the processor. When a response arrives and an open request is closed, the processor is then

unblocked. Seven different arrival rates are simulated for each type of video request condition (described below).

A Video-on-Demand server is normally subjected to two types of video request conditions. The first is the normal case where videos are evenly requested with no single video being in demand. Let's call this the balanced or evenly distributed video request condition. For the sample requests being simulated for the evenly distributed case, 6 nodes (nodes 2, 6, 8, 9, 11, 15) request different videos with blocks stored across the system.

We will see that since all nodes in the system are sharing the load, network performance under this condition is very good and leads to low queuing delays and low end-to-end latencies for most of the algorithms.

The second type of video request condition is the case where most requests are for a highly popular movie. In this case the nodes storing the blocks of this video become hotspots and tend to get congested. We refer to this as the hotspot video request condition. Figure 5.1 shows the data flow for the hotspot condition. To alleviate hotspots we assume that the video that is in demand is replicated and stored as a second video and requests for the video are split between the two copies. In this case 4 nodes are requesting 2 copies of the video with blocks localized in a particular area of the system.

We will show that since some nodes are doing most of the work in the system while others are idle, network performance in terms of queuing delays and end-to-end movie frame latencies is poorer than the previous case. However, the results for FAST are superior to Round-Robin and Stop-and-Go when handling such bursty traffic.

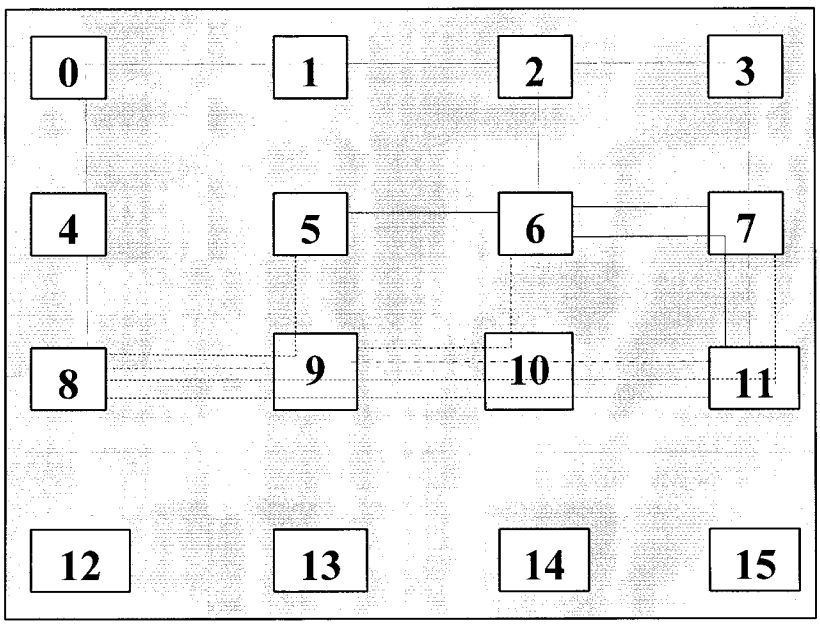


Figure 5.1: Hotspot movie request condition

5.3 Performance Metrics

This section provides the results of our simulation. The results have been grouped by the metric being studied. The major metrics we're interested in studying from the network and NIC point of view are:

- Throughput – Throughput in our case can be defined as the number of packets processed by the server during a specific interval. This metric measures the transmission rate of packets and provides an indication of burstiness of the traffic.

Throughput provides an indication of whether the system can handle the large amounts and bursty nature of video data.

- Queuing delay of packets – Packet queuing delay can be defined as the cumulative time a packet spends waiting in NIC input queues at each hop on its route. This metric is important since it measures the efficiency of the NIC in terms of queuing and scheduling.
- End-to-end latency of movie frames – End-to-end latency of a movie frame can be defined as the difference in time between when the frame is retrieved from memory, and, when the entire packet is received at the destination node. This metric is important since it measures the entire server's capability to efficiently utilize the system resources to get the frame quickly to its destination.
- Packet loss – Packet loss can be defined as the number of packets that are dropped during each run of the system due to lack of network resources - specifically buffers. This metric is important since it is useful in determining other measures of a multimedia server's performance such as miss ratio and jitter. Miss ratio and jitter are measured at the requesting client. Higher packet drops will lead to a larger miss ratio and jitter when playing the video at the client-end.

5.4 Throughput

We now study the throughput of the system in terms of the number of packets processed by the different algorithms under various loads.

As seen in Table 5, under the evenly distributed requests condition, Stop-and-Go (SG) only processes between 34,220 and 43,000 packets per 400,000 clock cycles. Round-Robin (RR) processes between 54,950 and 110,000 packets per 400,000 cycles. The 3 variations of FAST (T1 where time fame size is 1 cycle, T5 where time frame size is 5 cycles and T10 where time frame size is 10 cycles) process many more packets – between 54,950 and 195,500 per 400,000 cycles. In FAST, packets arrive at their destination at a much faster rate than Stop-and-Go, thus more effectively utilizing available network resources. Section 5.5 will also highlight this result.

	RR	SG	T1	T5	T10
Low Load	54,950	34,220	54,950	54,950	54,950
High Load	110,000	43,000	195,500	195,500	195,300

Table 5: Number of packets processed under evenly distributed request conditions

(Low load – 375 requests/1,000,000 cycles, High load – 2000 requests/1,000,000 cycles)

As seen in Table 6, under hotspot requests condition, Stop-and-Go processes the least number of packets about 12,600 across all loads. FAST processes the most packets - 43,950 at low loads and 102,000 at high loads thus better utilizing network resources by providing higher packet transmission rates.

	RR	SG	T1	T5	T10
Low Load	43,950	12,614	43,950	43,950	43,950
High Load	99,500	12,620	99,500	102,100	102,100

Table 6: Number of packets processed under hotspot request conditions

(Low load – 250 requests/1,000,000 cycles, high load – 1333 requests/1,000,000 cycles)

From Tables 5 and 6 we see that:

- FAST displays superior throughput with highest number of packets processed across all types of loads.
- Round-Robin displays fairly good throughput across all load conditions.
- Stop-and-Go has the poorest throughput. At higher loads, FAST processes 800% more packets than Stop-and-Go.

5.5 Latency Analysis

Now we study the end-to-end latency of video frames. The following results assume infinitely large queues and 0 packet loss. Each block of the movie being retrieved from memory is split into frames that are further divided into packets and flits. In this Section we study the end-to-end latency of a frame under various loads.

Figure 5.2 shows the latency for the algorithms being compared under various load levels and evenly distributed request conditions. We observe the following:

- Stop-and-Go's latency values are much worse than other algorithms including Round-Robin. This is as a result of Stop-and-Go's poor throughput (refer Table 4 in Section 5.4) and its strict admission policy where bursty packets are restricted from being transmitted and are made to wait at the ingress to the network.
- FAST provides consistently least latency values. FAST's latency values are about 200% better than Round-Robin and more than 500% better than Stop-and-Go.

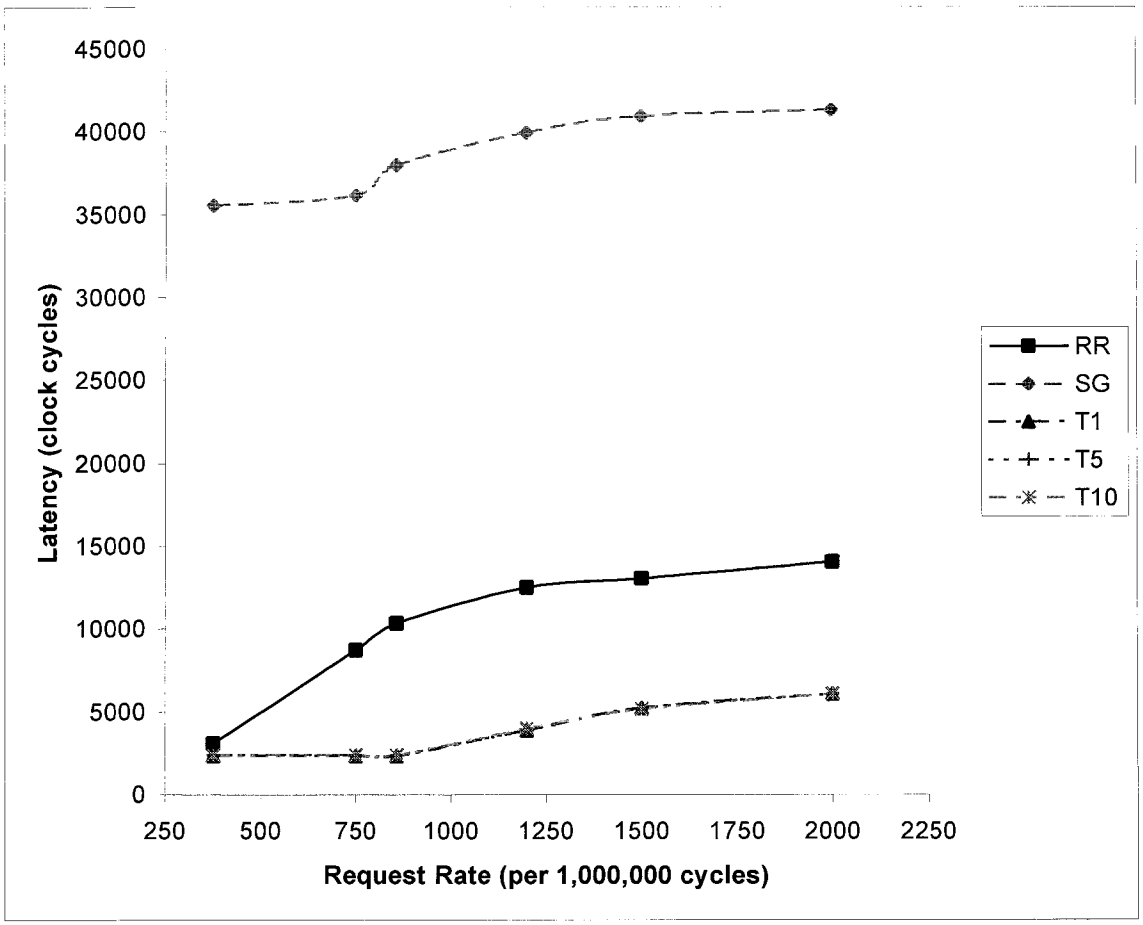


Figure 5.2: Latency comparison under evenly distributed request condition

Figure 5.3 compares T1, T5 and T10's latency values under various loads and for evenly distributed requests condition with infinitely large buffers. We observe that there is not much difference between the performances of the 3 variations of FAST. This is because under evenly distributed request conditions, congestion at nodes is minimal and the smoothness of traffic is maintained. Hence there is no significant advantage gained by delaying the packets at NICs in order to smooth out the traffic.

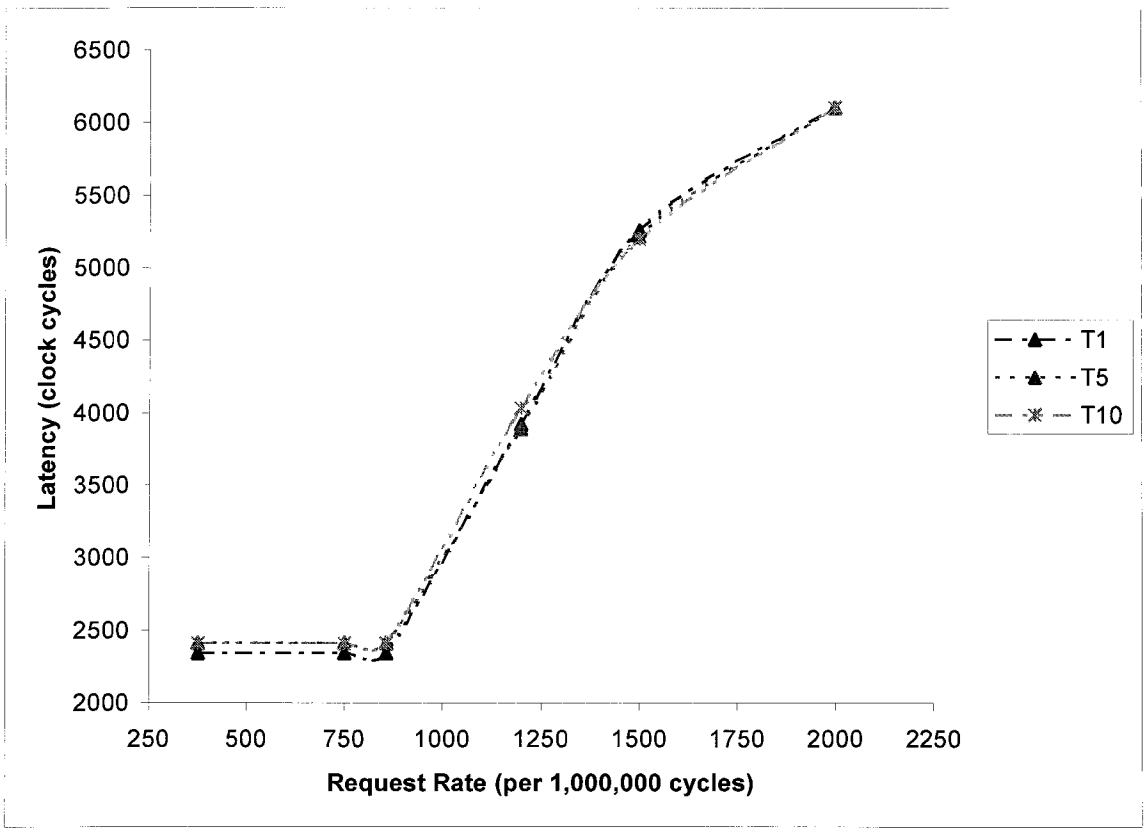


Figure 5.3: Latency comparison of FAST variations under evenly distributed request condition

Figure 5.4 shows the latency values for all algorithms under various load levels and hotspot request conditions. We observe that:

- Stop-and-Go's latency values are unacceptably high, again due to poor throughput (refer Table 5 in Section 5.4), leading to long wait times for packets that are ready to gain access to the network.
- The FAST algorithm provides the lowest latency values.

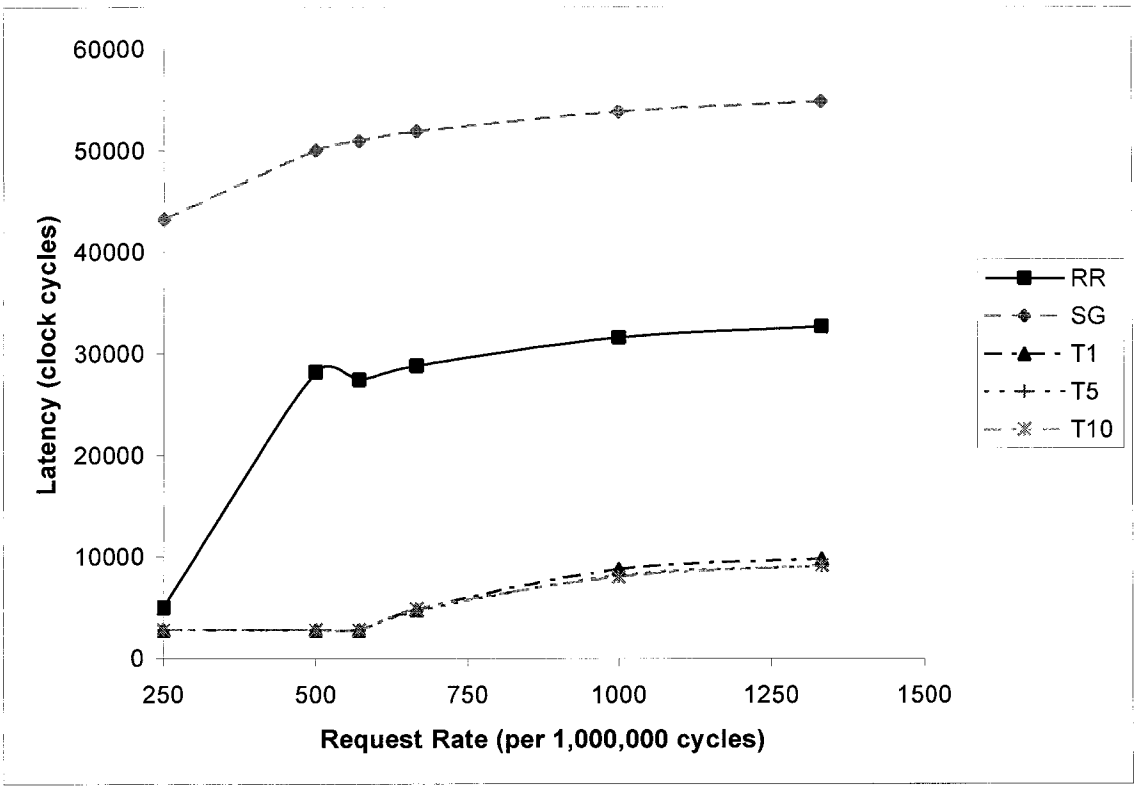


Figure 5.4: Latency comparison under hotspot request conditions

Figure 5.5 plots T1, T5 and T10's latency values for the hotspot request condition. From the figure we observe that:

- At lower loads, latency values for T1, T5 and T10 are comparable.
- As congestion and load increases, larger frame sizes T5 and T10 perform better than T1. This is because, at higher loads delaying packets longer at the NICs maintains the smoothness property of the traffic thus leading to lesser congestion, lower queuing delay and hence lower total latency.

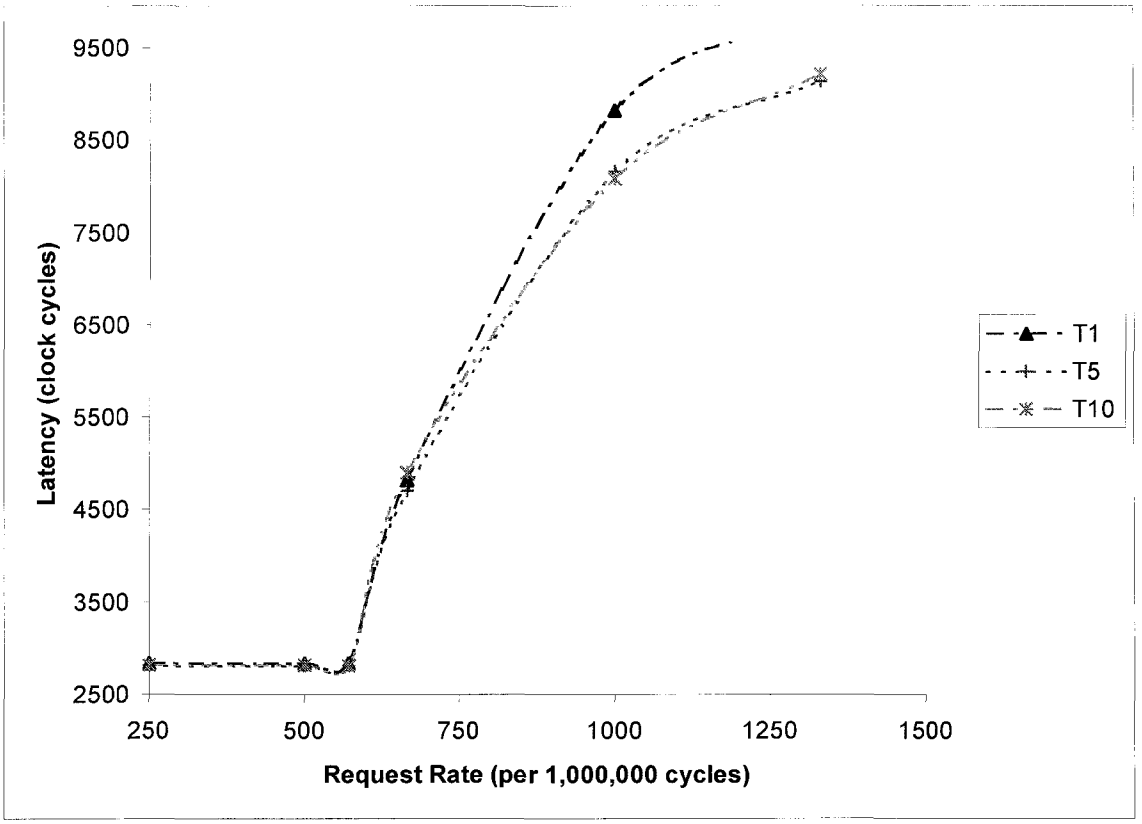


Figure 5.5: Latency comparison for FAST variations under hotspot request conditions

5.6 Queuing Delay Analysis

We now compare the queuing delays of Round-Robin, Stop-and-Go and FAST under various loads. We assume that all buffer queues are infinitely long and no packets are lost under any of the load conditions.

Figure 5.6 shows the queuing delay of Round-Robin, Stop-and-Go and the three variations of FAST (T1, T5, T10), when subjected to the evenly distributed movie requests condition with low request rate. We observe the following:

- Queuing delay for Round-Robin is very high when compared to other algorithms.

- Stop-and-Go has the best queuing delay, but displays lower throughput and higher end-to-end latency as seen in Sections 5.4 and 5.5. FAST's queuing delay while being higher than Stop-and-Go's, is still quite low.
- FAST's queuing delay is 600% better than Round-Robin.
- Under low loads there is no advantage to increasing time frame size in FAST. T1 has slightly lower queuing delay than T5 or T10.

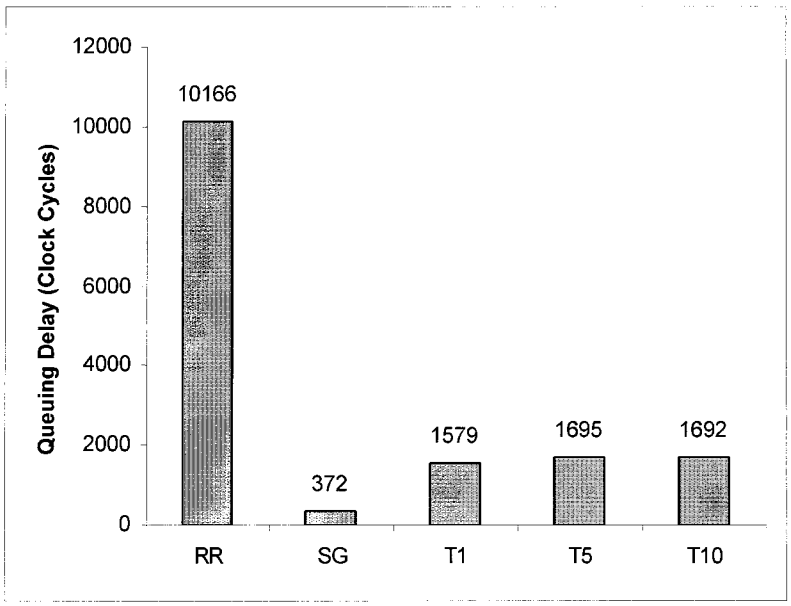


Figure 5.6: Packet queuing delay under low load and evenly distributed request condition

Figure 5.7 shows the queuing delay of all the algorithms when subjected to evenly distributed movie request condition at medium load (request rate). We observe the following:

- Queuing delay in FAST is 500% better than Round-Robin.
- Stop-and-Go still has the best queuing delay.
- Increasing time frame size has no major impact on queuing delay for FAST.

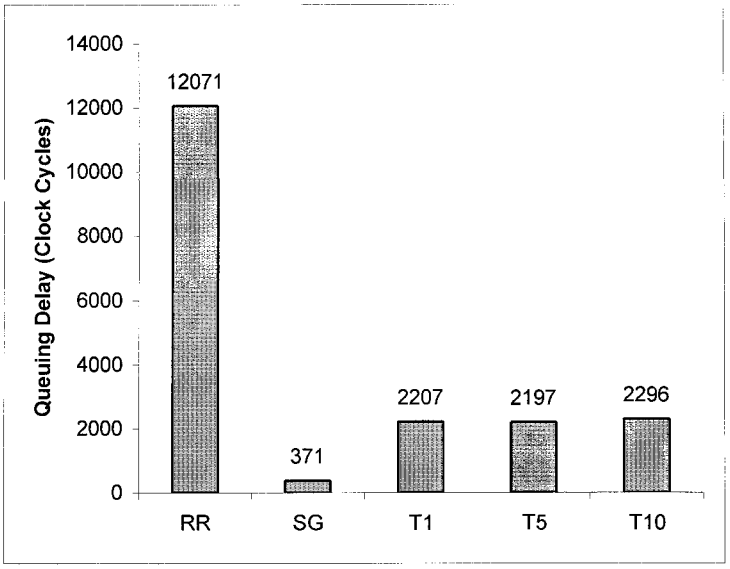


Figure 5.7: Packet queuing delay under medium load and evenly distributed request conditions

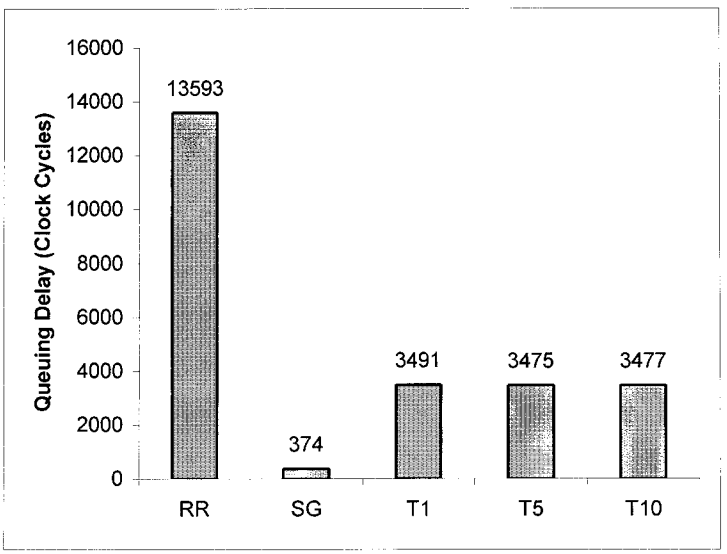


Figure 5.8: Packet queuing delay under high load and evenly distributed request conditions

Figure 5.8 shows the queuing delay of all the algorithms when subjected to the evenly distributed movie requests and high load or high request rate. From the figure we observe that:

- FAST again outperforms Round-Robin by a large margin. Stop-and-Go outperforms all the other algorithms.
- Once again increasing time frame size 'T' in FAST has no major impact on the queuing delay.

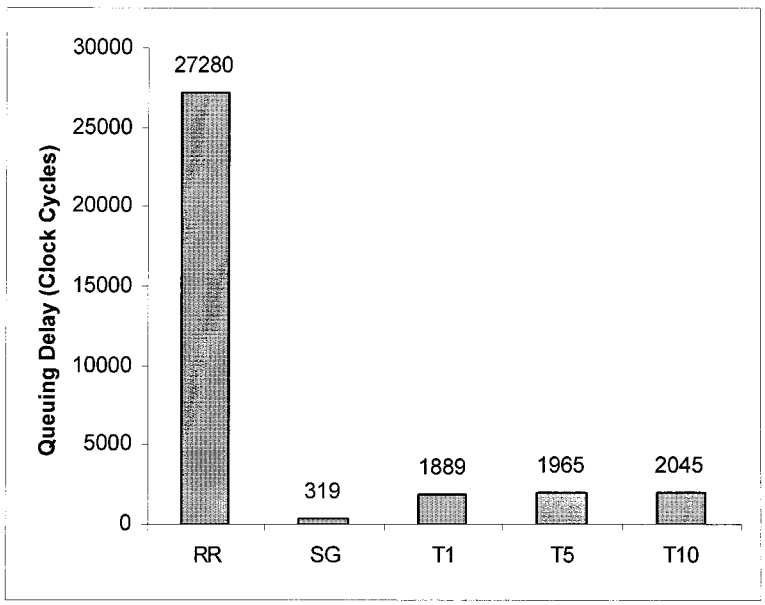


Figure 5.9: Packet queuing delay under low load and hotspot request conditions

Figure 5.9 examines the effect of hotspot condition and low load on queuing delay. We observe the following:

- Round-Robin's performance is very inferior, proving Round-Robin cannot manage congestion created due to the hotspots.
- Stop-and-Go's queuing delay is still very good and consistent with results obtained for the balanced movie requests condition.
- FAST's queuing delay although higher than Stop-and-Go's is still quite low.
- Once again increasing time frame size in FAST provides no real performance advantage.

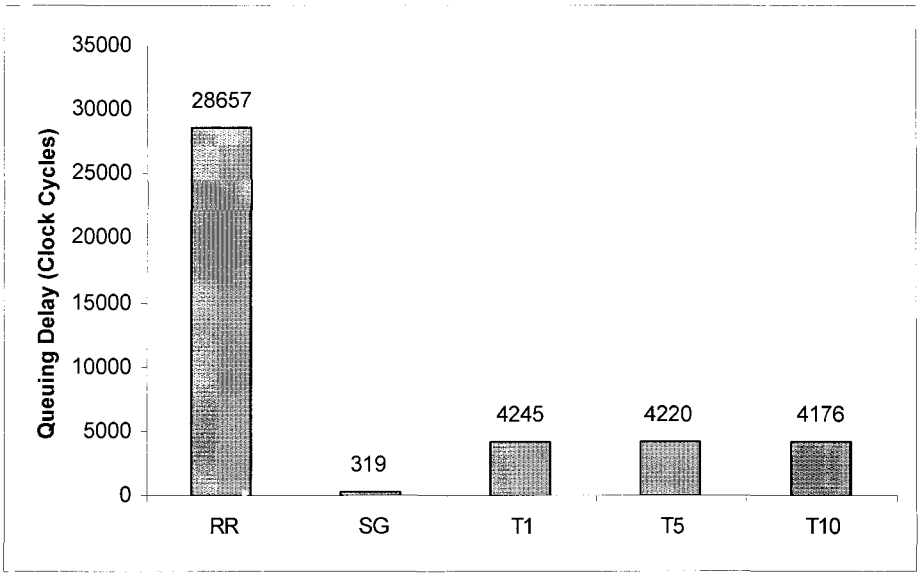


Figure 5.10: Packet queuing delay under medium load and hotspot request conditions

Figure 5.10 examines queuing delay under hotspot conditions and medium load. We see the following:

- The queuing delay displayed by Round-Robin is unacceptable.
- Stop-and-Go displays consistent queuing delays across all types of loads and request conditions examined so far.

Figure 5.11 depicts queuing delay under hotspot conditions and high load. The following are key observations:

- FAST's queuing delay is low and superior to Round-Robin.
- Stop-and-Go has been displaying consistently low queuing delay over different loads.
- Under heavier loads increasing time frame size 'T' now has an advantage in FAST. We see that queuing delay for T5 and T10 is approximately 10% better than T1's queuing delay.

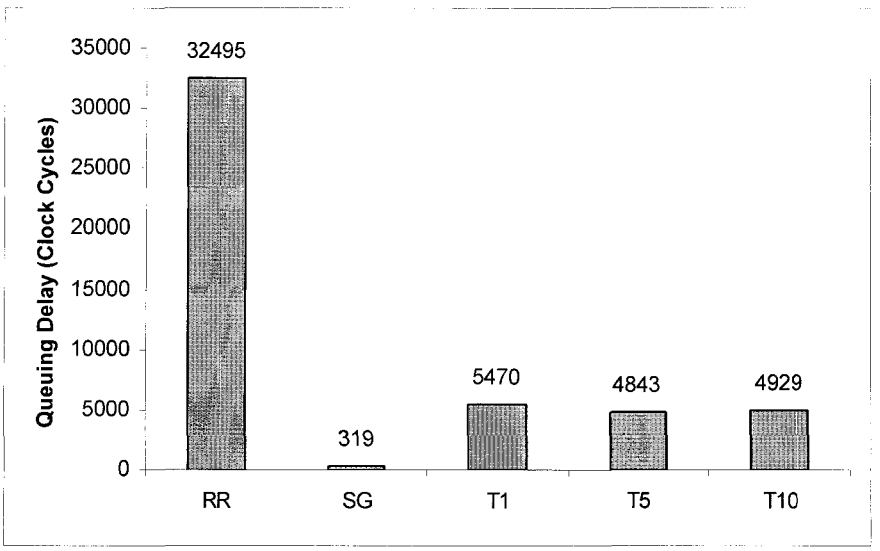


Figure 5.11: Packet queuing delay under high load and hotspot movie request conditions

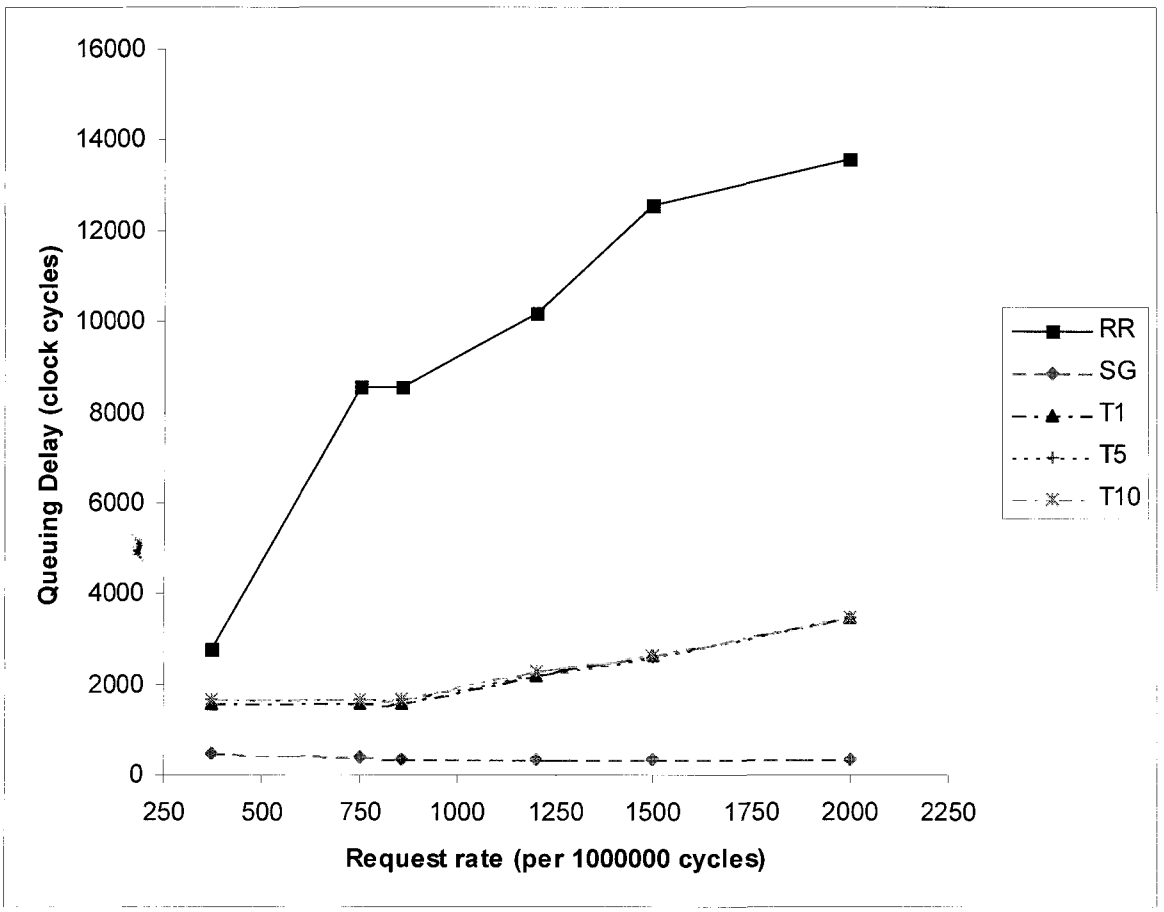


Figure 5.12: Queuing delay comparison under evenly distributed request conditions

Figure 5.12 summarizes the results obtained for the evenly distributed condition across various loads. Although all three FAST algorithms (T1, T5 and T10) outperform Round-Robin, they do not match Stop-and-Go's low and constant delay bounds. This is due to the relaxed admission control policy of FAST allowing bursty traffic to enter the network. Stop-and-Go, however, restricts bursty traffic leading to under-utilization of network resources (some connections do not fully utilize the time slots reserved for them). As seen before, Stop-and-Go's throughput is very low indicating that at any given time, very few packets exist in the network. Hence very few packets get queued in the input queues of NICs thus leading to lower queuing delays.

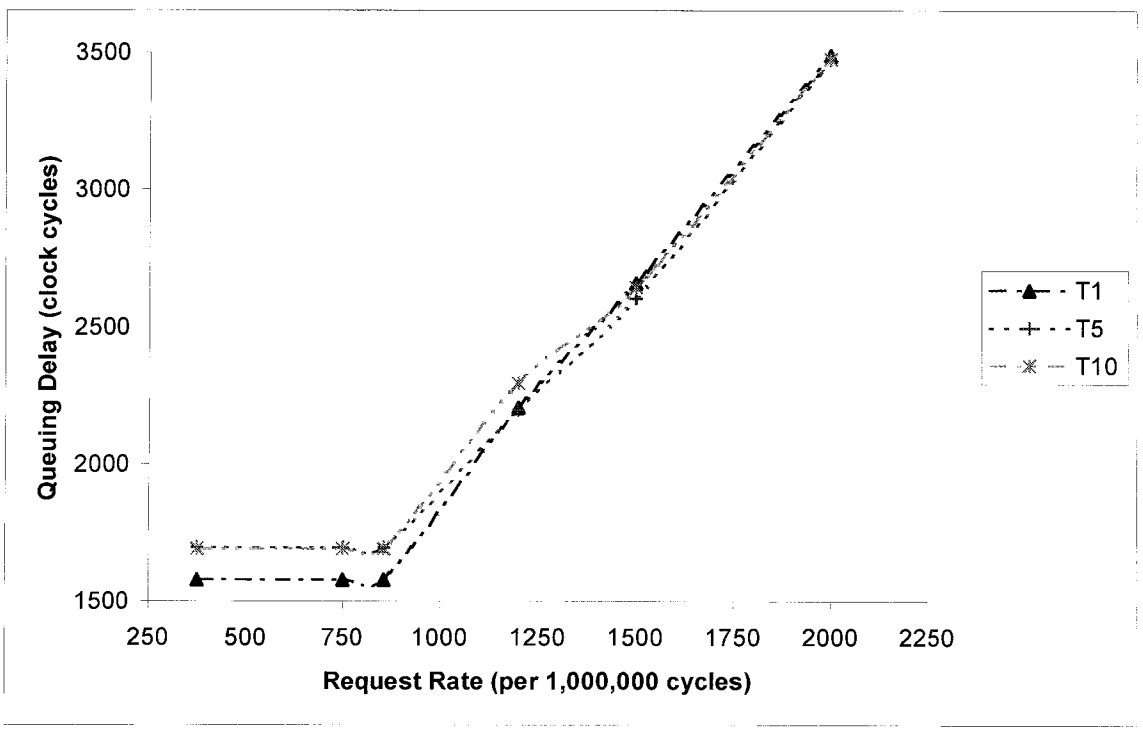


Figure 5.13: Queuing delay of FAST under evenly distributed request conditions

Figure 5.13 compares the three variations of FAST – T1, T5 and T10. From the figure we observe that at low to medium loads, lower time frame size provides better queuing delays. Under evenly distributed conditions, increasing the time frame size does not have an advantage.

Figure 5.14 summarizes the results obtained for the hotspot condition across various loads. Once again Round-Robin has the worst performance, while Stop-and-Go has the lowest delays. FAST however shows consistent performance with low delays (when compared to Round-Robin), despite allowing bursty traffic into the network.

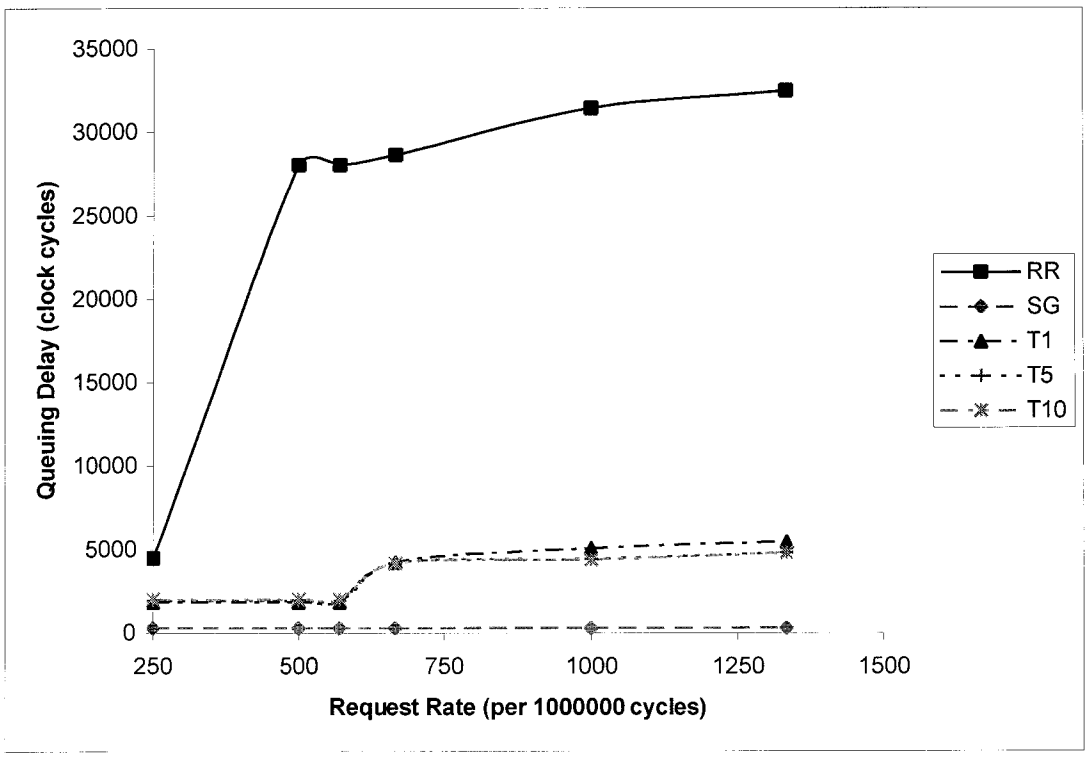


Figure 5.14: Queuing delay comparison under hotspot request condition

Figure 5.15 compares the three FAST algorithms under hotspot conditions. In the figure, the sharp rise in queuing delay at medium loads and then the flattening out at higher loads

is because of the kicking in of the request control mechanism, which allows only a certain number of open requests per node (refer Section 5.2). We observe that under low loads T1 performs slightly better than T5 and T10. However, as load increases, T5 and T10 outperform T1. This is because at larger 'T' values, we delay packets longer at NICs. At lower loads this additional delay only worsens queuing delay. However, at higher loads when congestion starts to become a problem, this additional delay helps to smooth out the traffic thus reducing congestion.

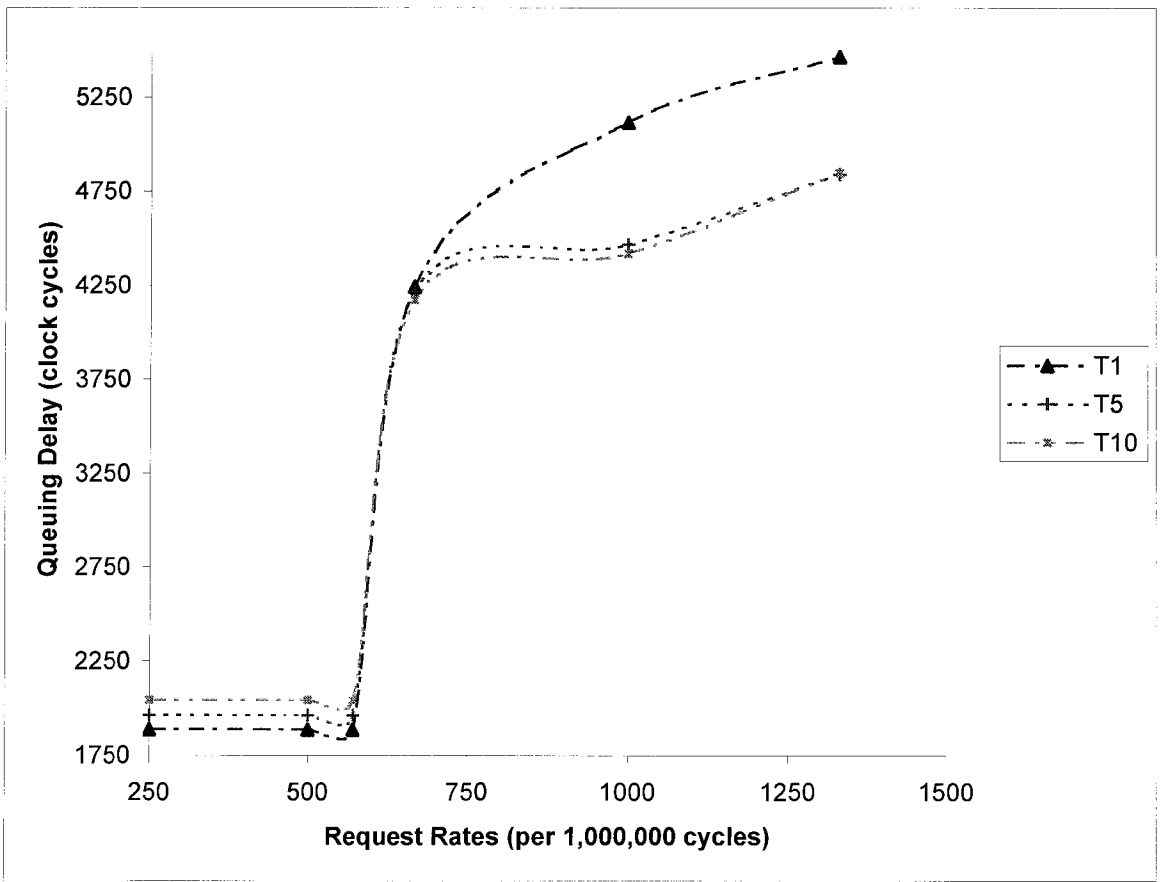


Figure 5.15: Queuing delay of FAST under hotspot request conditions

Figure 5.16 compares the queuing delay of FAST under the two types of movie request conditions, namely, evenly distributed and hotspot conditions. We see that as expected

queuing delay under hotspot condition is higher than that under evenly distributed condition. This is because of higher load on a subset of nodes in the hotspot request condition, as opposed to balanced load on all nodes in the evenly distributed request condition.

In both cases at lower loads, T1 performs marginally better than T5 and T10. However, in the hotspot case, as load and subsequently congestion increases, T1 becomes significantly worse than T5 and T10. This is because larger frame size helps to alleviate congestion. T5 and T10 while being slightly worse than T1 at lower loads, provide more consistent results than T1 across all loads.

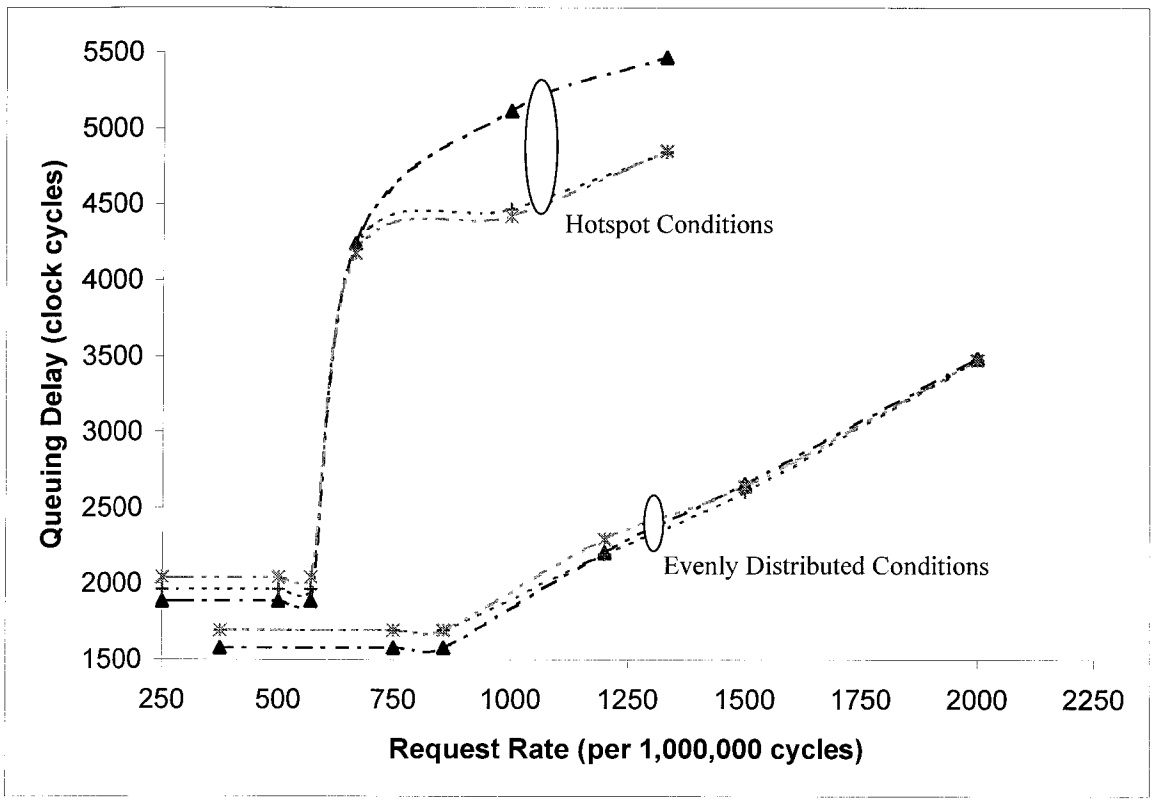


Figure 5.16: Comparison of evenly distributed and hotspot conditions

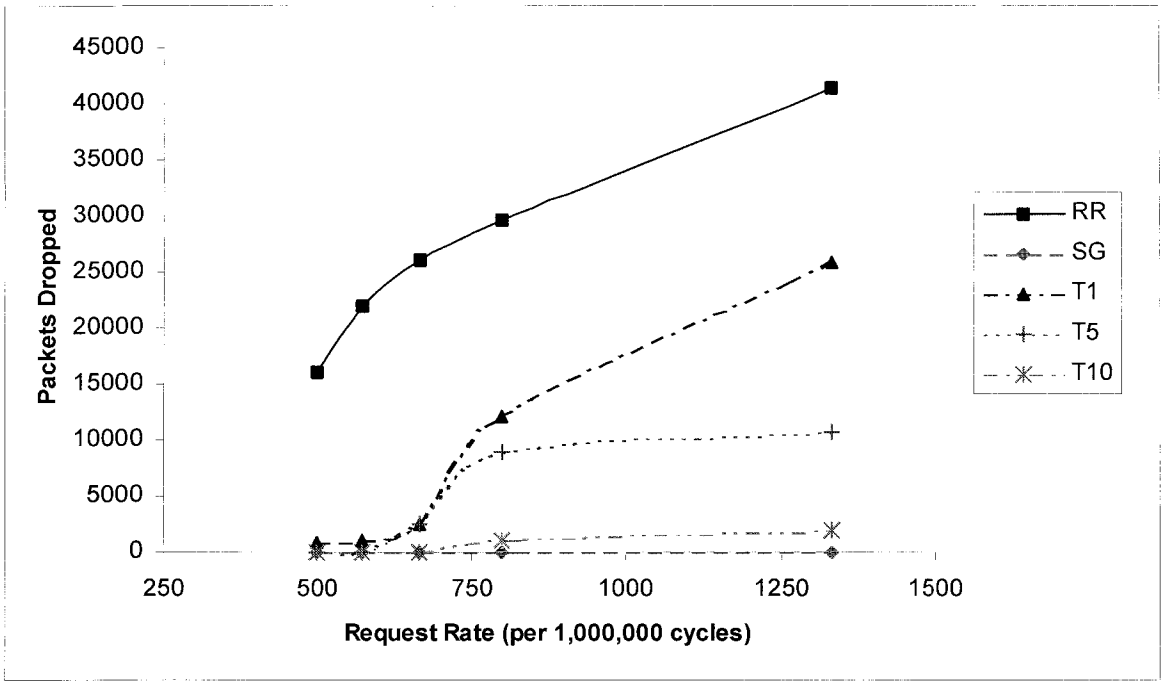
5.7 Packet Loss Analysis

Packet loss is an important metric as it gives an indication of the jitter during playback at the client. After observing the infinite buffers case, where there were no packets dropped, we now reduce the number of buffers in the input queues to 3000 (96 KB). We will formulate the results for this finite buffers case and then add DVCs to the system and observe the difference this makes to the packet loss.

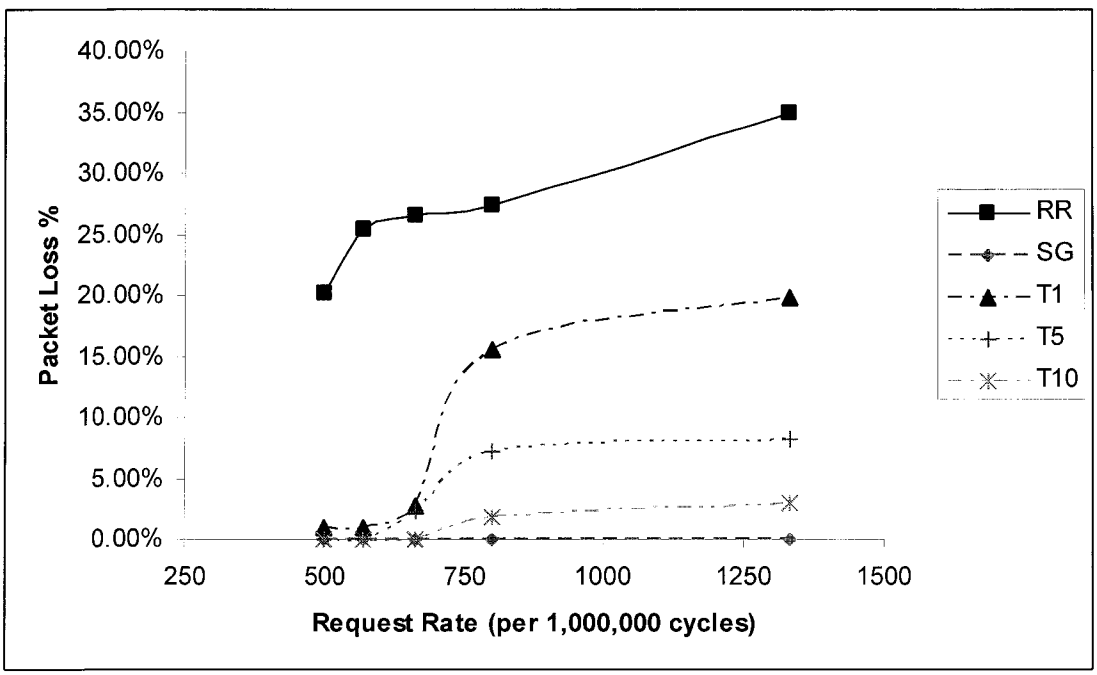
5.7.1 Absence of DVCs

Figure 5.17 compares the number of packets dropped for each algorithm with queue sizes of 96 KB. Stop-and-Go does not drop any packets since it processes very few packets, (refer Table 5) at all loads. However there is some packet loss associated with FAST. It is important to notice that at bigger time frame sizes – namely T10, the number of packets dropped becomes low, when compared to Round-Robin or even T1. At low loads, T10 does not drop any packets. At medium to high loads, the packet drop count is still very low. This compares favorably with Stop-and-Go.

Round-Robin's packet drop count even at low loads is too high to make it a feasible algorithm for Video-on-Demand servers unless the queue sizes are very large.



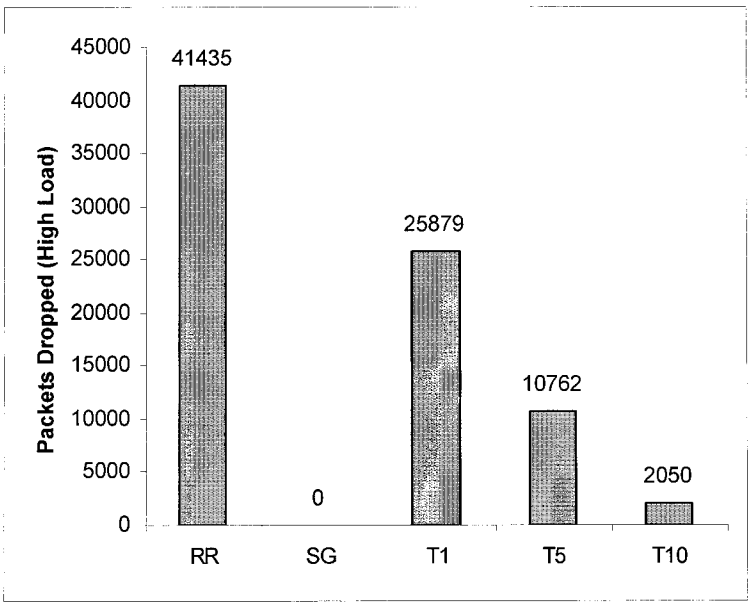
(a): Packet drop count



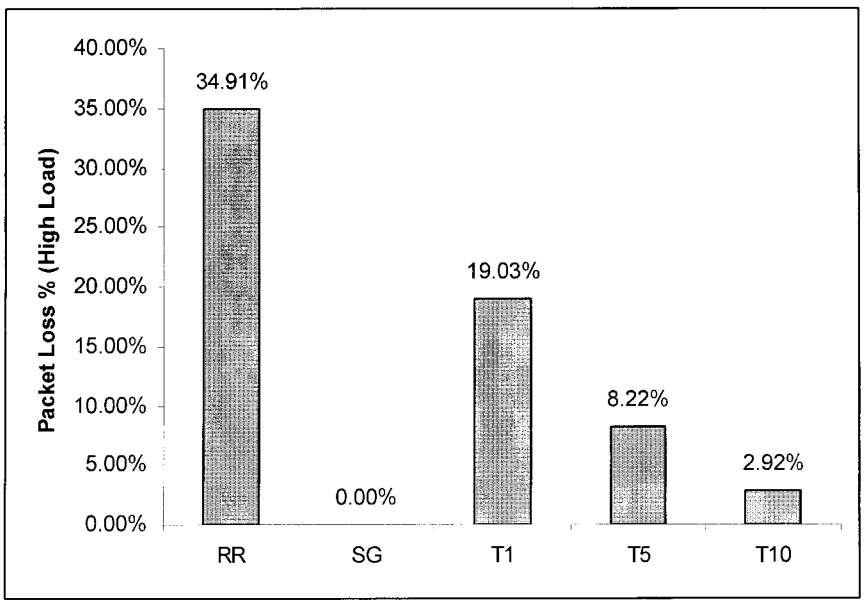
(b) Percentage of packets lost

Figure 5.17: Packet loss with finite buffers and in the absence of DVCs

Figure 5.18 depicts the packet drop counts at high load. Even at such high loads and under hotspot conditions, T10 drops less than 3% of the packets processed.



(a) Packet drop count



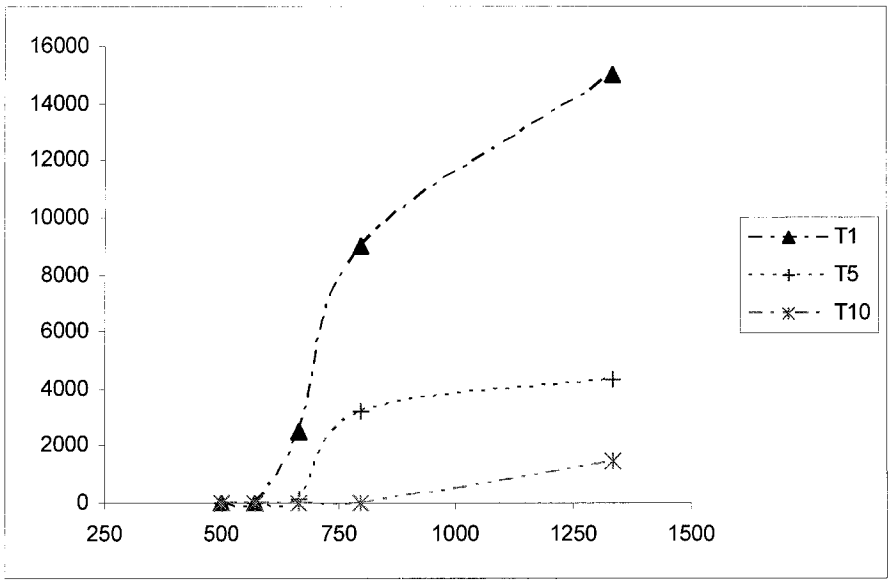
(b) Percentage of packets lost

Figure 5.18: Packet loss at high load (1333 requests per 1,000,000 clock cycles) and in the absence of DVCs

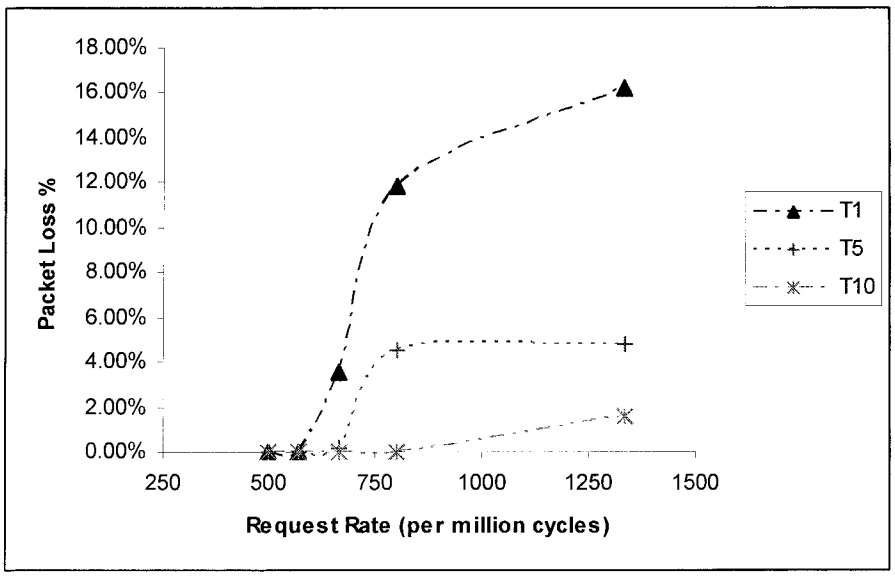
In the next section, we will see that the impact of addition of DVCs to FAST.

5.7.2 Presence of DVCs

Since Round-Robin and Stop-and-Go do not use DVCs, this section presents results only for FAST.



(a) Packet drop count



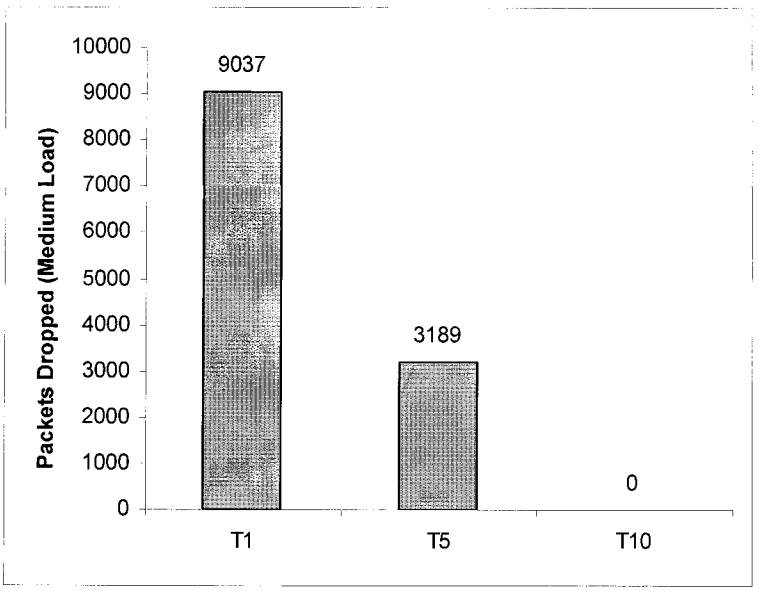
(b) Percentage of packets lost

Figure 5.19: Packet loss with finite buffers and in the presence of DVCs

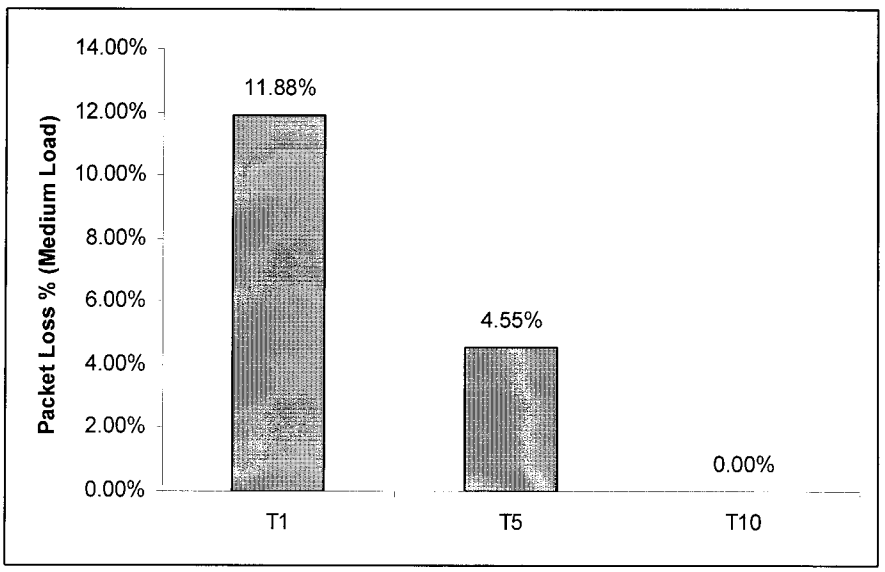
Figure 5.19 compares the three variations of FAST– T1, T5 and T10 with the same total buffer size at each node (25 * 96KB = 2,400KB), but with DVCs in addition to SVCs. The total buffer size is divided equally between the 45 virtual channels (25 SVCs and 20 DVCs), such that each queue is the same length and has approximately 53KB of buffer space. The DVC queue size is the same as the SVC queue size. From the figure we see that:

- At lower frame sizes, namely T1, packets dropped are still high but have reduced by 60% compared to the non-DVC case.
- With larger sized frames, namely T10, the improvement by adding DVCs is more than 70%.
- With larger sized frames (T5 and T10), packets are dropped only at high loads. For low and medium loads there are no packets dropped.

Figure 5.20 shows the packet drop counts for the three variations of FAST at medium load. Notice the difference in number of packets dropped just by increasing the time frame size 'T'. At larger 'T' values, namely T10, there are no packets dropped. This is comparable to Stop-and-Go.



(a): Packet drop count

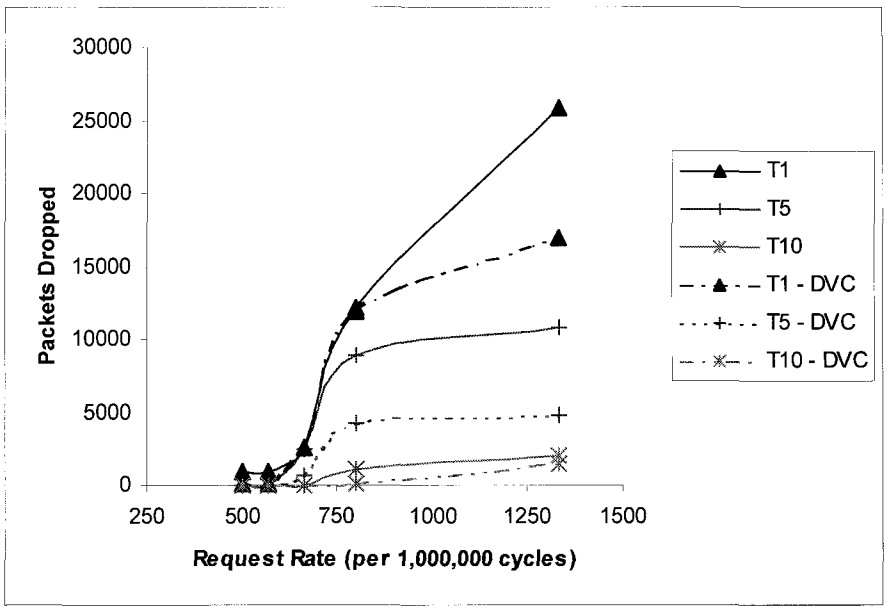


(b) Percentage of packets lost

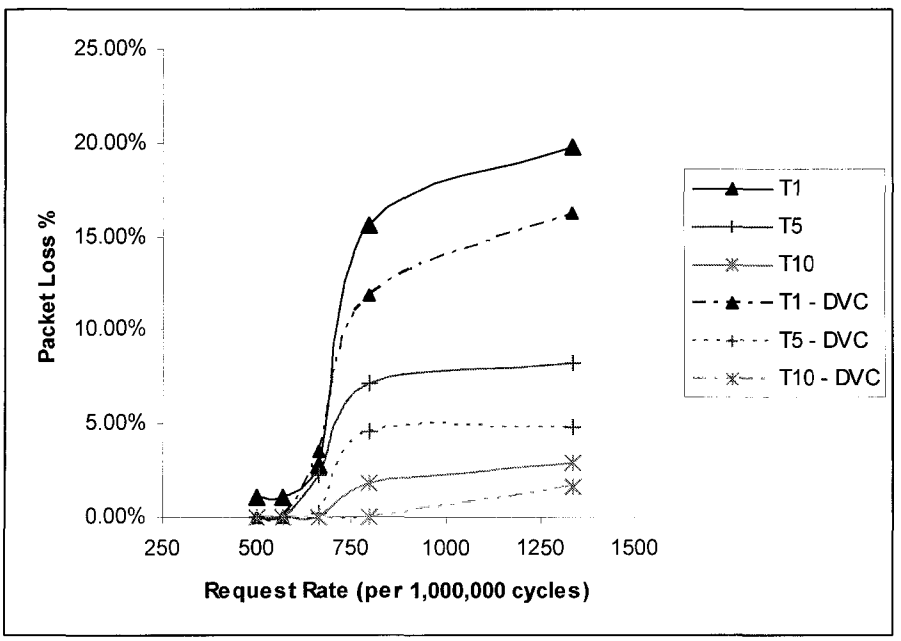
Figure 5.20: Packet loss at medium load (800 requests per 1,000,000 clock cycles) and in the presence of DVCs

Figure 5.21 compares the packet drop counts of the three variations of FAST in the absence of DVCs and in the presence of DVCs under various loads. In each case, the

improvement in packets dropped when DVCs are used is significant. This is especially noticeable as the load increases.

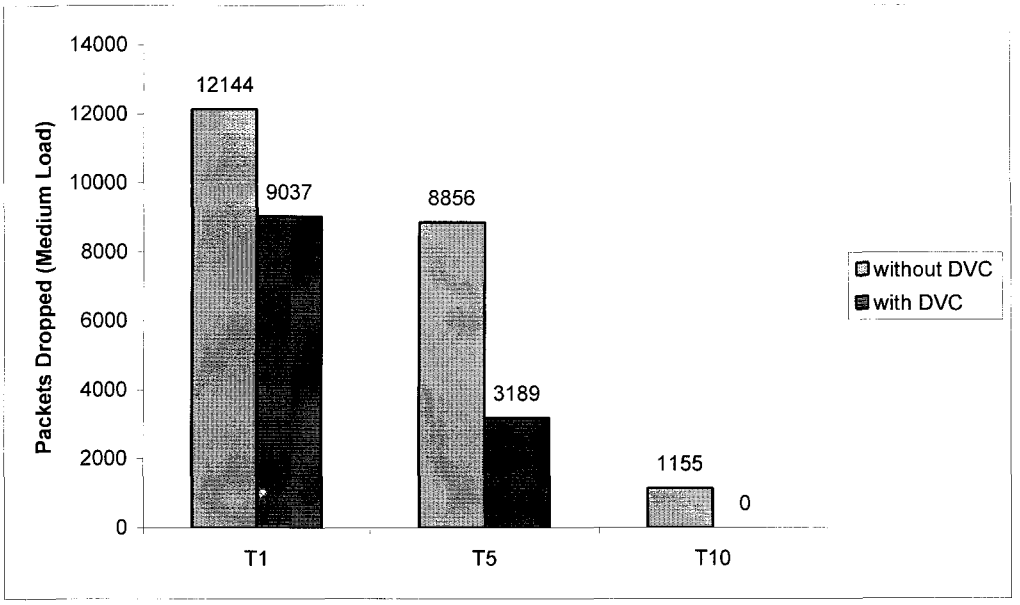


(a) Packet drop count

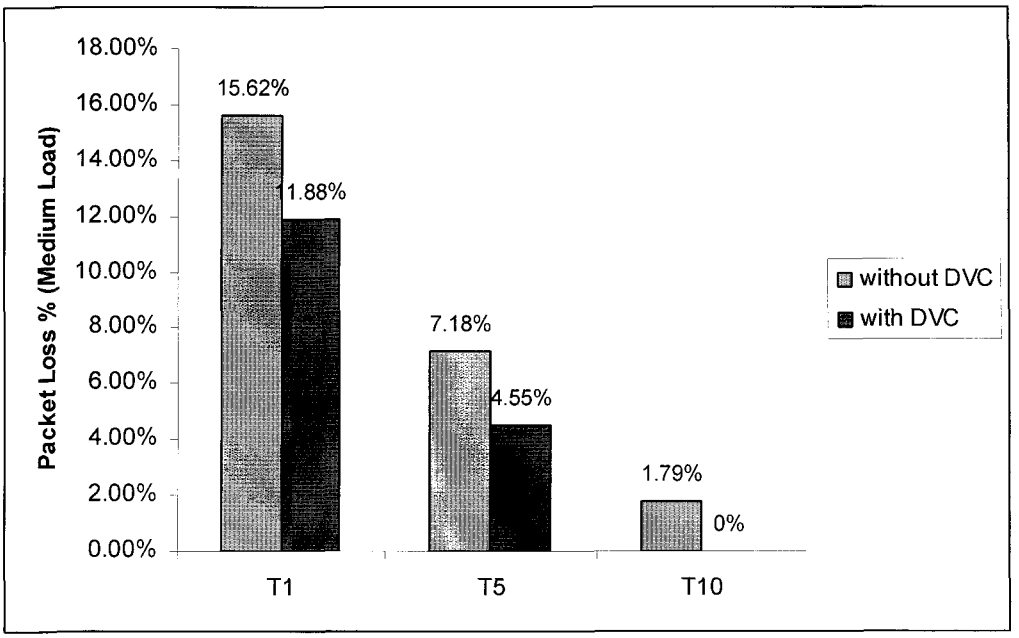


(b) Percentage of packets lost

Figure 5.21: Comparison of packet loss in the absence and in the presence of DVCs



(a) Packet drop count



(b) Percentage of packets lost

Figure 5.22: Comparison of packet loss in the presence and absence of DVCs and at a medium load (800 requests per 1,000,000 clock cycles)

Figure 5.22 compares the packets dropped values of the three variations of FAST in the absence of DVCs and in the presence of DVCs. The comparison is shown for a medium load. All three algorithms T1, T5 and T10 perform better when they use DVCs even though total buffer space at each node remains the same as when only SVCs were used. This is because the additional DVC queues allow for faster processing of eligible packets and also create an illusion of more buffer space. When switching input packets to output channels, the packets in the SVC and the DVC are compared and the more eligible packet is switched. If the packet at the head of the SVC is not eligible while the packet at the head of the DVC is eligible, then this packet is switched. While in the non-DVC case, if the packet at the head of the SVC queue is not eligible, then none of the packets in that queue are switched.

We also note from the figure that packet loss values for larger time frame sizes (T10) are better than the packet loss values for smaller time frame sizes (T1 and T5). We can conclude that using FAST with a larger time frame size, namely T10, leads to good system performance at all load levels.

5.8 Performance of FAST in the Presence of DVCs

We now analyze how FAST performs with a finite number of buffers and in the presence of DVCs, as compared to its performance when it has infinite buffers. We consider only the hotspot condition at high load, since this is the worst-case that we have simulated. In order to do a fair comparison, we need to choose a queue size that results in 0 packet loss, since there are no packets lost in the infinite buffer case. A SVC queue size of 7,500

buffers (240 KB) when used in conjunction with DVCs of the same size that are allocated as required, gives us 0 packet loss.

Figure 5.23 shows the queuing delay under these conditions for all the three variations of FAST. We note that, as before, T5 and T10 perform better than T1 by about 10%, since larger time frames manage congestion better at high loads. Round-Robin and Stop-and-Go are not shown in this Section, since they do not use DVCs.

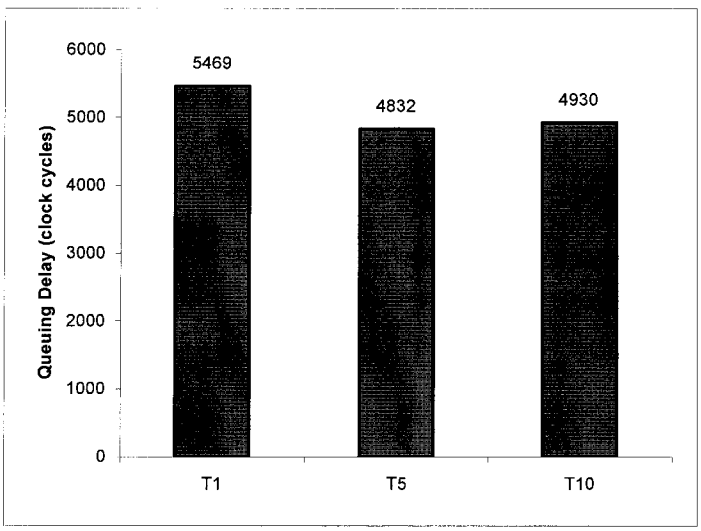


Figure 5.23: Queuing delay of FAST algorithms at high load (1333 requests per 1,000,000 clock cycles) and in the presence of DVCs

Figure 5.24 compares the queuing delay of the FAST algorithms using infinite buffers with the queuing delay using finite buffers and DVCs. We see that the performance of both cases under hotspot condition and high load is almost identical. There is a negligible difference that can be ignored since these are simulated conditions.

This similarity leads us to conclude that using DVCs is similar to adding to the size of the SVCs. An appropriate size of DVC that will lead to minimum or no packets lost is just like using infinite buffers for the queues.

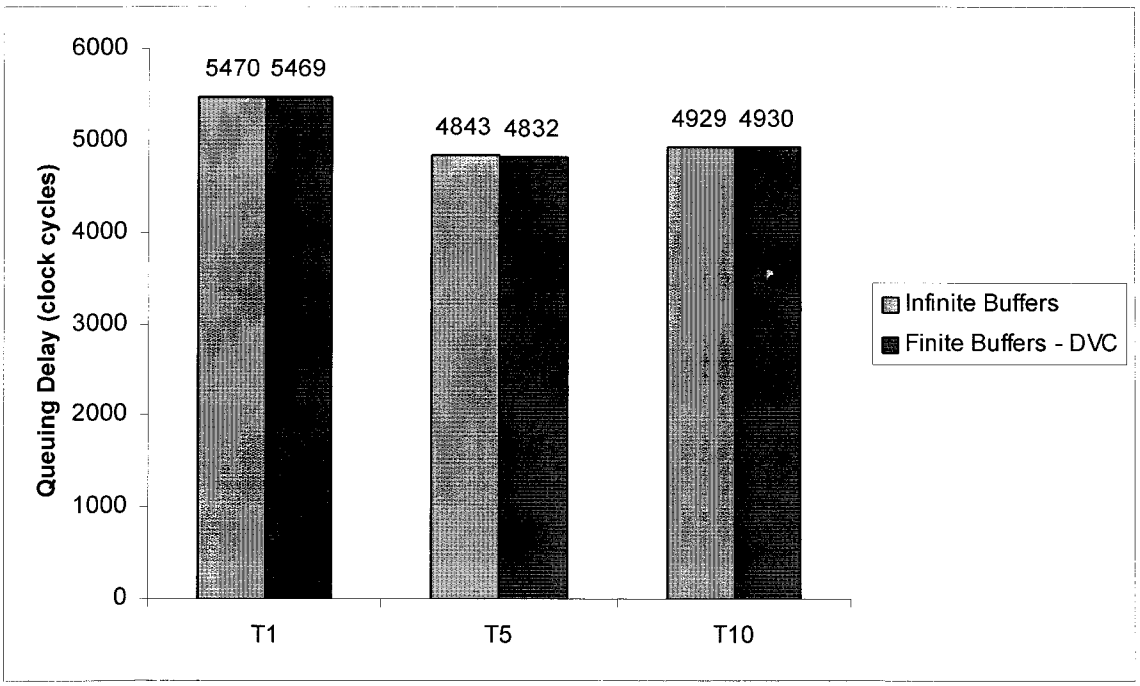


Figure 5.24: Comparison of queuing delay using infinite buffers with queuing delay using finite buffers and DVCs at high load (1333 requests per 1,000,000 clock cycles)

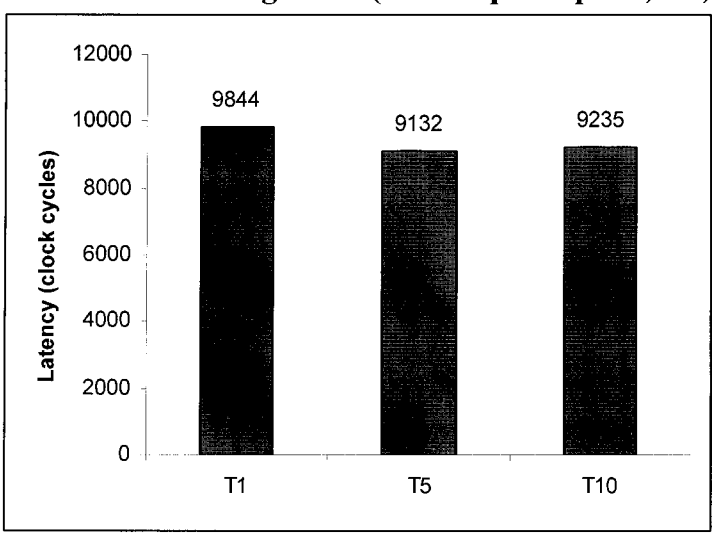


Figure 5.25: Latency of FAST algorithms at high load (1333 requests per 1,000,000 clock cycles) and in the presence of DVCs

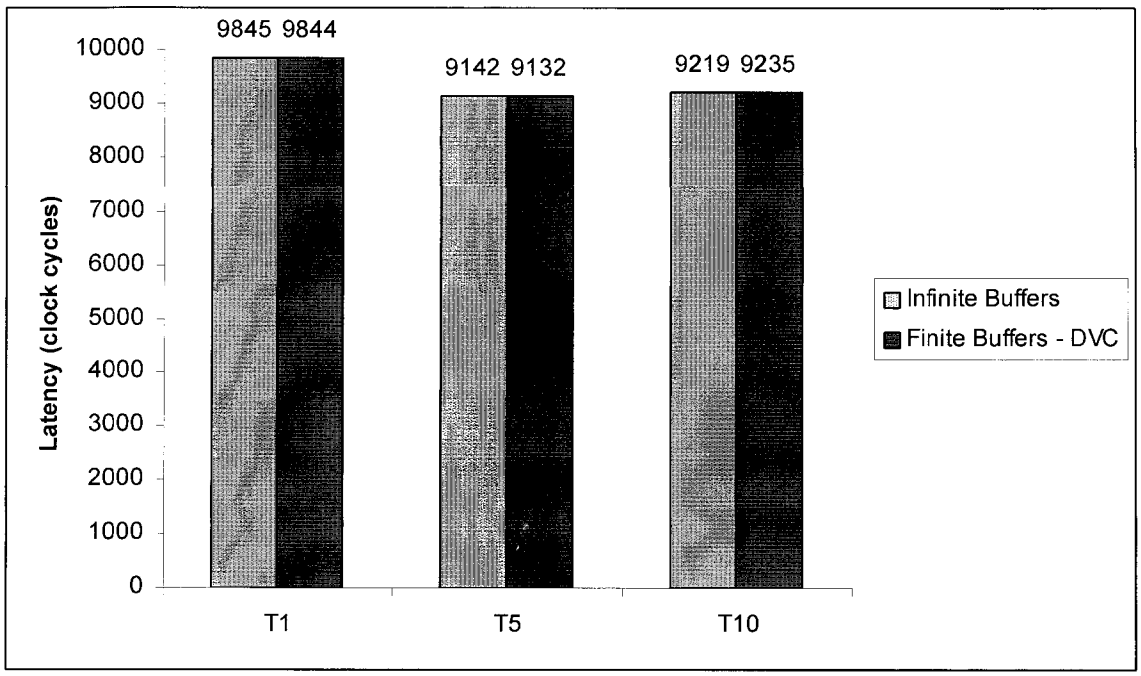


Figure 5.26: Comparison of latency at high load (1333 requests per 1,000,000 clock cycles) using infinite buffers with latency using finite buffers and DVCs

Figure 5.25 and Figure 5.26 do a similar study for latency of the movie frames. The results obtained are similar to that of the queuing delay. Namely –

- Larger time frame sizes perform better under high loads.
- Running the system with finite buffers that lead to minimum packet loss, gives us identical performance as running the system with infinite or very large buffers.

5.9 Buffer Analysis

In this section we perform buffer analysis based on the busiest queue of the most congested or bottlenecked node in a 4x4 mesh network. Results are collected for the worst-case or hotspot video request condition. These results help us to calculate an appropriate queue size required to handle video traffic with minimum packet loss.

5.9.1 Bottleneck Analysis

For the hotspot condition, we need to identify which NIC is the bottleneck causing the worst queuing delays. For the example chosen, we have 4 nodes – 2, 6, 8 and 9 – that make movie requests M1 and M2 that result in response blocks being sent from nodes 6, 8, 11 and nodes 5, 7, 11 respectively. This is shown in Figure 5.27. The coloured nodes are the ones making the request and the patterned lines link them to the responding nodes.

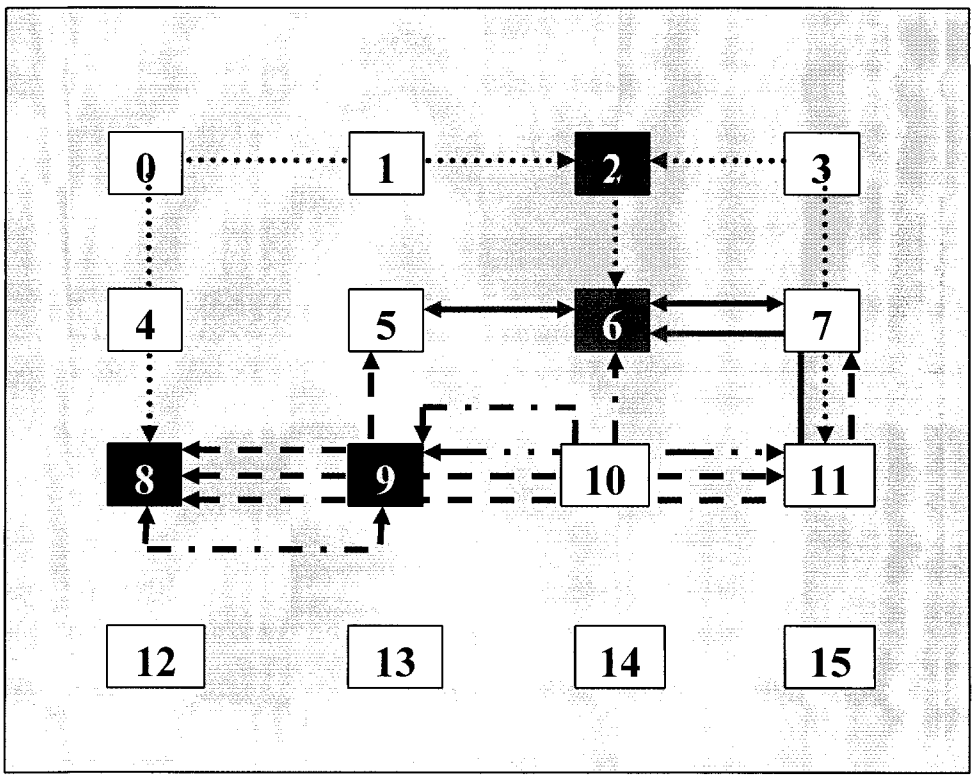


Figure 5.27: Data flow under hotspot conditions

From the figure we can see that NIC 9 seems to be the busiest since it is making requests to nodes 6, 8 and 11 for movie blocks, while serving as an intermediate or transit node for all movie blocks headed to node 8. Hence, NIC 9 would serve as the best candidate for worst-case buffer analysis.

In order to identify the busiest queue in NIC 9, let us examine Figure 5.28. This figure shows all queues that service request and response packets originating from node 9, transiting through node 9 or terminating at node 9, for our hotspot condition example.

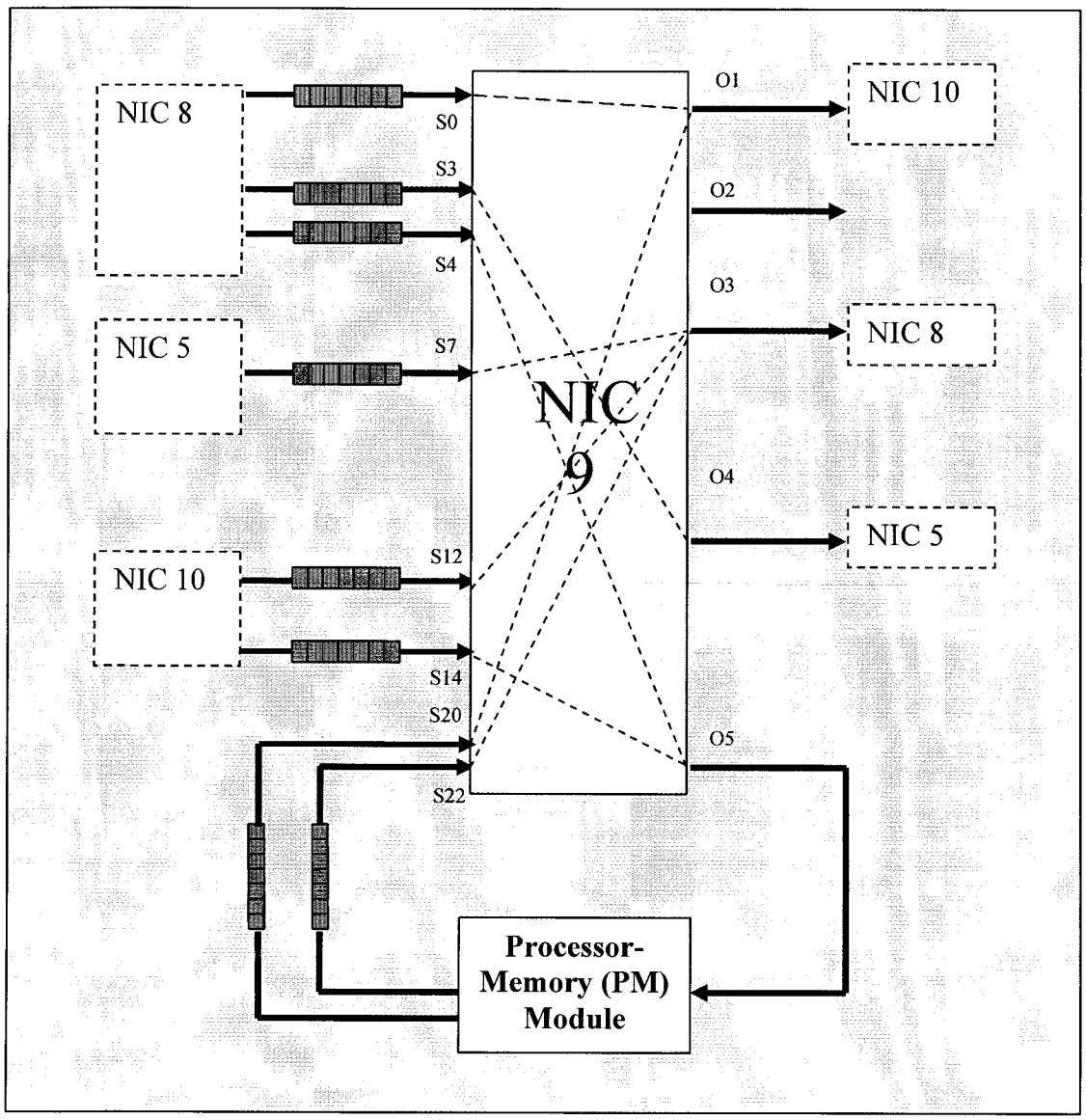


Figure 5.28: Queues utilized in bottleneck NIC under hotspot conditions

Queues S20 and S22 store request packets for the movie requests that node 9 is making to nodes 6, 8, and 11. Since request data size is very small compared to response data size, these two queues are not very long. Queues S4 and S14 store response packets that

terminate at NIC 9. Local node 9 quickly consumes these packets, and PM9 is not busy responding to any requests.

The remaining queues S0, S3, S7, S12 all store packets transiting through NIC 9. Of these S0 stores request packets going from node 8 to node 10, S3 stores request packets going from node 8 to node 5, S7 stores response packets going from node 5 to node 8, and S12 stores response packets going from node 10 to node 8. Queues S0 and S3 are not very busy since they store smaller sized request data. Response data route between node 5 and node 8 is not very busy, hence we can eliminate queue S7 as being our culprit. This means that queue S12 should be the busiest since it stores large-sized response data coming from nodes 7 and 11 and transiting through node 9 on their way to node 8. Hence we conclude that queue S12 in NIC 9 should be the most congested queue in the system. Experimental results ratify this conclusion as seen in Tables 7 and 8.

Request Rate (per 1,000,000 cycles)	S0 Avg	S3 Avg	S4 Avg	S7 Avg	S12 Avg	S14 Avg	S20 Avg	S22 Avg
500	0	0	54	2	**692**	13	0	0
571	0	0	62	2	**791**	15	0	0
667	0	0	3	10	**5957**	2	0	76
1000	0	0	1	4	**6118**	1	0	79
1333	0	0	1	4	**7459**	1	0	79

Table 7: Average queue lengths for busiest NIC under hotspot conditions

Request Rate (per 1,000,000 cycles)	S0 Max	S3 Max	S4 Max	S7 Max	S12 Max	S14 Max	S20 Max	S22 Max
500	7	4	354	9	**1609**	204	19	8
571	7	4	354	9	**1609**	204	19	8
667	8	4	564	155	**7882**	466	20	84
1000	7	4	354	18	**8743**	204	23	84
1333	7	4	354	18	**9943**	204	23	84

Table 8: Maximum queue lengths for busiest NIC under hotspot conditions

5.9.2 Average and Maximum Queue Lengths

Figure 5.29 shows the average and maximum queue lengths at different request rates for the bottleneck queue S12 in NIC 9.

We observe from the figure that at high load (1333 requests per 1,000,000 cycles), the queue length is the maximum. The queue is made up of flit-size buffers.

Hence,

Maximum length of queue S12 = 9943

Flit size = 256 bits

Queue Size = 9943 * 256 bits = 2545408 bits = 319KB.

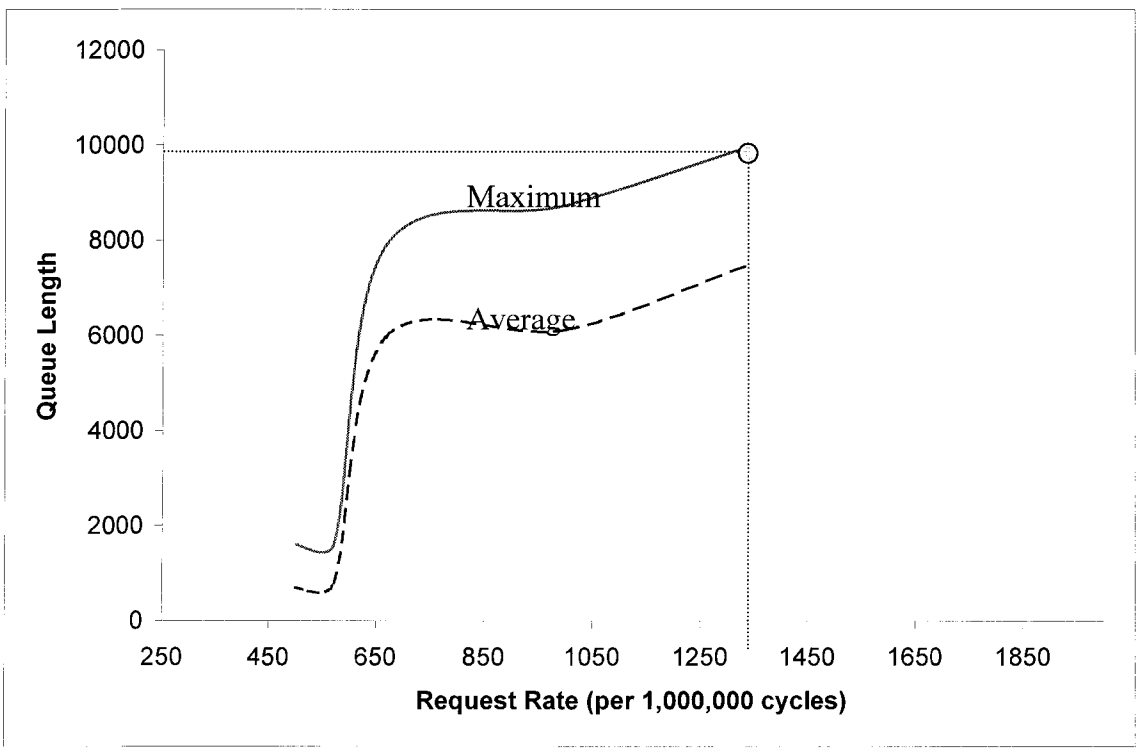


Figure 5.29: Average and maximum queue lengths for bottleneck queue

Figure 5.30 shows the cumulative average and maximum queue lengths for the bottlenecked NIC 9.

We observe from the figure that at any given time, the maximum total memory usage of the node for all its queues S0 to S24 does not exceed 10,500.

Hence,

NIC 9's memory usage at high load = 10,500 * 256 bits = 2688000 bits = 336 KB.

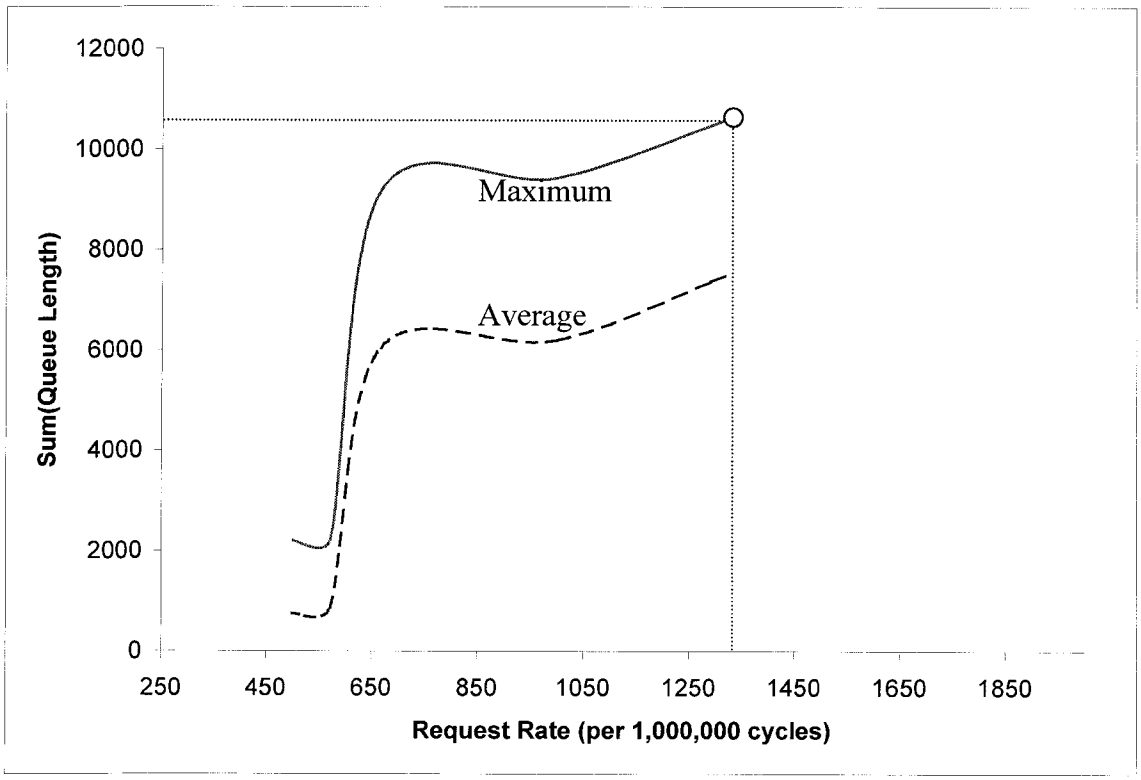


Figure 5.30: Average and maximum queue lengths for bottlenecked node

5.9.3 Cumulative Buffer Usage Distribution

Figure 5.31 plots the probability distribution for buffer size when the system is running at high load under hotspot conditions.

It may be argued that the average length (approximately 262 KB) may be a sufficient queue length for the NIC queues in the system. However, from the distribution between the average and maximum cases, we can see that running the system with the average queue length would not be sufficient almost half the time in the case of the bottleneck queue. This is because the probability that queue length would be less than 262 KB at any given time was only 0.54. This implies that 46% of the time, queue length was greater than the average. Therefore, the bottleneck queue requires being at maximum length (approximately 319 KB) at all times in order to avoid packet drops. Since the bottleneck queue is the busiest queue, its maximum length (319 KB) is the worst-case queue length. Making all the NIC queues this length would allow for sufficient buffer space at each NIC.

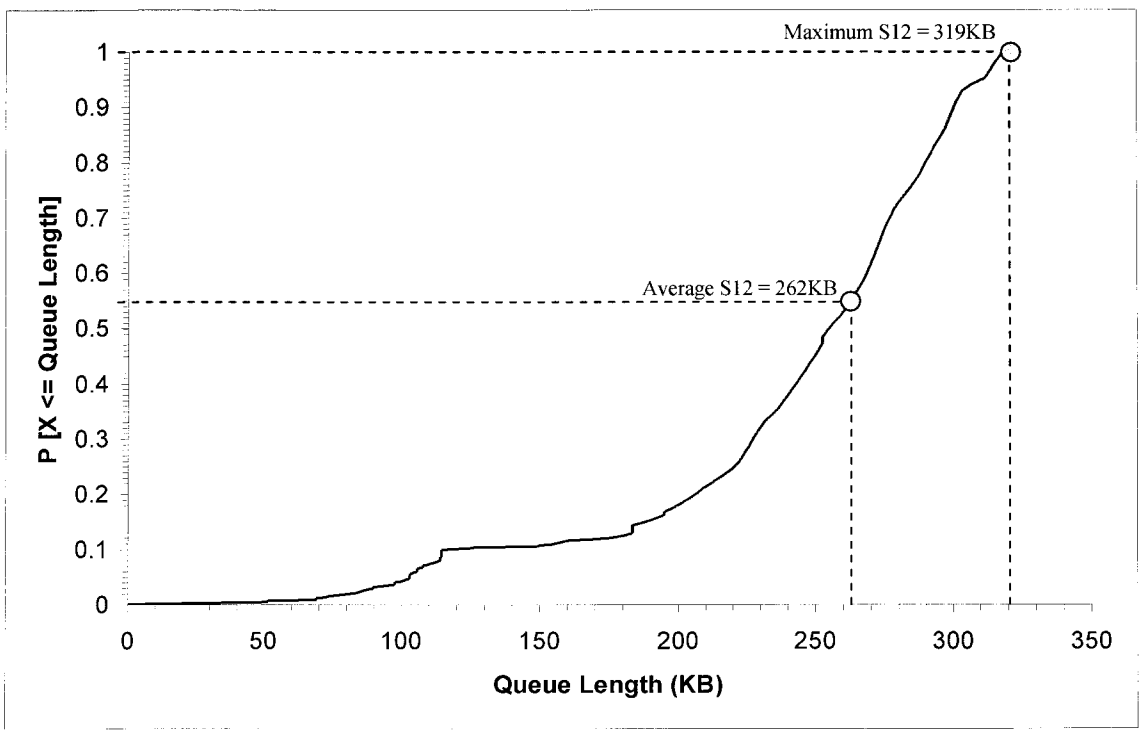


Figure 5.31: Cumulative distribution of buffer usage under high load and hotspot conditions

Summary

This chapter explored the results of the simulation. Major metrics like queuing delay, latency and packet drop counts were compared for 3 algorithms – Round-Robin, Stop-and-Go and FAST (with 3 variations - namely, T1, T5 and T10). The following were the findings:

- Round-Robin's queuing delay and packet drop counts were the worst of all the algorithms studied
- Stop-and-Go had the least queuing delay and packet drop counts, but at the cost of very poor end-to-end latency and throughput
- FAST had the most consistent queuing delay, latency and packet drop counts under all types of loads
- FAST provided the highest throughput and the best performance despite allowing burstiness into the network

In order to support real-time multimedia traffic, the selected algorithm should provide a very consistent end-to-end latency and superior throughput. With the above mentioned findings, it can be concluded that FAST is more suitable for supporting real-time multimedia traffic than Round-Robin and Stop-and-Go.

Of the three FAST variations studied, T10's performance was comparable to T1 and T5 at low loads but at high loads and in the presence of congestion, T10 provided better results. In the presence of finite number of buffers, T10 has the lowest number of packet drops. With the addition of DVCs, T10 outperformed T1 and T5 with no packets dropped

at low and medium loads and very few packets dropped at high loads. This made T10 the ideal choice for use in Video-on-Demand servers.

Addition of DVCs to the system with low and appropriate SVC and DVC size led to minimal packet loss and similar network performance as using infinite buffers.

Therefore, a system that uses FAST queuing and scheduling, a larger time frame size like T10 with a central pool of buffers (DVCs), exhibits good performance even in the presence of hotspots, congestion and high loads.

In addition, we also provided buffer analysis results for better memory utilization and optimizing resources. We observed that supporting real-time multimedia traffic with delay bounds is memory intensive and buffer engineering is an important issue to consider.

Chapter 6

Conclusions and Future Work

In this thesis, we proposed a comprehensive architecture for a Video-on-Demand system with special emphasis on network and communications issues. We designed a 2D multiprocessor Video-on-Demand server. As part of the NIC design, we proposed new flow control and packet scheduling mechanisms. In particular, we introduced an efficient, new algorithm called Frame-based Arbitration and Scheduling Technique (FAST), designed to handle real-time multimedia video traffic.

Performance of the proposed FAST algorithm was compared to other known algorithms, namely Round-Robin and Stop-and-Go. We proved that FAST has better throughput and end-to-end latency, the main performance measures to support real-time multimedia traffic in Video-on-Demand systems. Using simulation, we showed that FAST's policy of allowing bursty data into the network leads to superior throughput and end-to-end latency for video data for various loads. The addition of dynamic virtual channels to the FAST algorithm further improved system performance. The low and consistent queuing delay and packet loss figures also emphasized that FAST is well suited for handling video data traffic in multiprocessor based Video-on-Demand systems.

We clearly established the following results to prove that the proposed FAST algorithm is better in handling real-time multimedia traffic compared to Round-Robin and Stop-and-Go, for average and hot spot conditions:

- FAST had the most consistent throughput, total latency, queuing delay and packet drop count combination under all types of loads. In particular, FAST had 300% less packet queuing delays when compared to Round-Robin.
- FAST provided the highest throughput and the best performance despite allowing burstiness into the network.
- FAST is more capable of handling bursty hotspot conditions than Round-Robin and Stop-and-Go.
- FAST provided lower end-to-end latency and higher throughput than Round-Robin and Stop-and-Go. In particular, FAST had at least 200% lower end-to-end latency than Round-Robin and 500% lower latency than Stop-and-Go for the studied cases.

The work described in this dissertation can be extended in the following ways:

- Feedback control loop to the source nodes can be added to incorporate admission control and to improve service guarantees.
- FAST can be compared to other scheduling algorithms, such as FCFS, virtual clock, and RCQ.
- Proposed architecture can be prototyped for commercialization.

References

[1] Lionel M. Ni, Philip K. McKinley, A Survey of Wormhole Routing Techniques in Direct Networks, *IEEE Computer*, vol. 26, pp. 62-76, February 1993.

[2] Khalid M. Al-Tawil, Mostafa Abd-El-Barr, Farooq Ashraf, A Survey and Comparison of Wormhole Routing Techniques in Mesh Networks, *IEEE Network*, pp. 38-45, March/April 1997.

[3] Po-Chi Hu, Leonard Kleinrock, An Analytical Model for Wormhole Routing with Finite Size Input Buffers, *15th Intl. Teletraffic Congress*, pp. 549-560, Washington DC, June 1997.

[4] Jeffrey T. Draper, Joydeep Ghosh, A Comprehensive Analytical Model for Wormhole Routing in Multicomputer Systems, *Journal of Parallel and Distributed Computing*, vol. 23, pp. 202-214, November 1994.

[5] Rajendra V. Bopanna, Suresh Chalasani, Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks, *IEEE Trans. on Computers*, vol. 44, no. 7, pp. 848-863, July 1995.

[6] William J. Dally, Virtual-Channel Flow Control, *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, March 1992.

[7] William J. Dally, Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, *IEEE Trans. on Computers*, vol. C-36, no. 5, pp. 547-553, May 1987.

[8] Lixia Zhang, Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks, *ACM Trans. on Computing Systems*, vol. 9, no. 2, pp. 101-124, September 1990.

[9] S. Jamaloddin Golestani, Congestion-Free Transmission of Real-Time Traffic in Packet Networks, In *Proc. IEEE INFOCOM*, pp. 527-542, San Francisco, California, June 1990.

[10] S. Jamaloddin Golestani, A Stop-and-Go Queuing Framework for Congestion Management, In *Proc. ACM SIGCOMM Symp. Communications Architecture and Protocols*, pp. 8-18, Philadelphia, Pennsylvania, September 1990.

[11] Hui Zhang, Srinivasan Keshav, Comparison of Rate-Based Service Disciplines, In *Proc. ACM SIGCOMM*, pp. 113-121, August 1991.

[12] Shobana Balakrishnan, A Priority-Driven Flow Control Mechanism for Real-Time Traffic in Multiprocessor Networks, *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 7, pp. 664-678, July 1998.

[13] Jae H. Kim, Andrew A. Chien, Rotating Combined Queuing (RCQ): Bandwidth and Latency Guarantees in Low-Cost, High-Performance Networks, In *Proc. 23rd Intl. Symp. on Computer Architecture*, May 1996.

[14] Jack Y.B. Lee, Parallel Video Servers: A Tutorial, *IEEE Multimedia*, vol. 5, no. 2, pp. 20-28, June 1998.

[15] D. James Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, Multimedia Storage Servers: A Tutorial and Survey, *IEEE Computer*, vol. 28, no. 5, pp. 40-49, May 1995.

[16] John G. Apostolopoulos, Wai-tian Tan, Susie J. Wee, *Video Streaming: Concepts, Algorithms and Systems*, Hewlett-Packard Company, September 2002.

[17] R. Tewari, R. Mukherjee, D. Dias and H. Vin, Design and Performance Tradeoffs in Clustered Multimedia Servers, In *Proc. 3rd IEEE Intl. Conf. on Multimedia Computing Systems*, pp. 144-150, Tokyo, Japan, May 1996.

[18] Min-You Wu, Wei Shu, Optimal Scheduling for Parallel CBR Video Servers, In *Proc. Multimedia Tools and Applications*, vol. 14, no. 1, pp. 79-99, May 2001.

[19] Joseph Kee-Ying Ng and Shuhua Xiong, The Design and Implementation of Video Servers in Video-on-Demand Systems, *Parallel and Distributed Computing Practices*, vol. 2, no. 3, September1999.

[20] Govindan Ravindran, *Performance Issues in the Design of Hierarchical-ring and Direct Networks for Shared-Memory Multiprcessors*, Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Toronto, December 1997.

[21] M. H. MacDougall, *Simulating Computer Systems: Techniques and Tools*, Cambridge, MA: MIT Press, 1987.

[22] D. Jadav, C. Srinilta, A. Choudhary, and P. B. Berra, Techniques for Scheduling I/O in a High Performance Multimedia-on-Demand Server, *Journal of Parallel and Distributed Computing*, vol. 30, no. 1, pp. 190-203, November 1995.

[23] A. L. N. Reddy, Scheduling and Data Distribution in a Multiprocessor Video Server, In *Proc. Intl. Conf. on Multimedia Computing and Systems*, pp. 256-263, May 1995.

[24] M. M. Buddhikot, G. M. Parulkar, and J. R. Cox Jr., Design of a Large Scale Multimedia Storage Sever, *Journal of Computer Networks and ISDN Systems*, pp. 504-524, December 1994.

[25] W.J. Bolosky et al, The Tiger Video Fileserver, In *Proc. NOSSDAV 96*, April 1996.

[26] Dapeng Wu Hou, Y.T. Wenwu Zhu Ya-Qin Zhang Peha, J.M., Streaming Video over the Internet: Approaches and Directions, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 282-300, March 2001.

[27] Gregory J. Conklin et al., Video Coding for Streaming Media Delivery on the Internet, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 269-281, March 2001.

[28] Chia-Wen Lin, Jeongnam Youn, Jian Zhou, Ming-Ting Sun, Iraj Sodagar, MPEG Video Streaming with VCR Functionality, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 415-425, March 2001.

[29] Dapeng Wu Hou, Y.T. Wenwu Zhu Hung-Ju Lee Tihao Chiang Ya-Qin Zhang Chao, H.J., On End-to-End Architecture for Transporting MPEG-4 Video over the Internet, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, no. 6, pp.923-941, September 2000.

[30] Web-site for Moving Picture Experts Group - http://www.MPEG.org

[31] Seth Abraham, *Issues in the architecture of direct interconnection schemes for multiprocessors*, University of Illinois at Urbana-Champaign, Champaign, IL, March 1990.

[32] A. Agarwal, Limits on Interconnection Network Performance, *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398-412, October 1991.

[33] Ralph Duncan, A Survey of Parallel Computer Architectures, *IEEE Computer*, vol. 23, pp. 5-16, February 1990.

[34] Sivarama P. Dandamudi, Reducing Hot-Spot Contention in Shared Memory Multiprocessor Systems, *IEEE Concurrency,* pp. 48-59, January-March 1999.

[35] Kalman M., Steinbach E. Girod B., Adaptive media playout for low-delay video streaming over error-prone channels, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, no. 6, pp. 841-851, June 2004.

[36] Wiegand T., Sullivan G.J, Bjontegaard G., Luthra A., Overview of the H.264/AVC video coding standard, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, July 2003.

[37] Izadi B.A., Ozguner F., Enhanced cluster k-ary n-cube, a fault-tolerant multiprocessor, *IEEE Trans. Computers*, vol. 52, no. 11, pp. 1443 – 1453, November 2003.