TRAFFIC ENGINEERING METHODS FOR A NETWORK PROCESSOR BASED MULTIMEDIA ROUTER

By

Amr Elramly

B.Sc. Electrical and Computer Engineering, Ein Shams University, 1982

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

Electrical and Computer Engineering

THE UNIVERSITY OF BRITISH COLUMBIA

July 2006

ABSTRACT

The thesis proposes methods to build network processor based multimedia router that provides more services than what is offered by the most expensive network routers. The new methods provide scalability to serve hundreds of thousands of multimedia streams and serve fast enough to meet the low latency required by the multimedia application. The first method proposed is a proxy agent that turns the router into multimedia aware router by preparing the profiling and the routing information for the forwarder and the scheduler, which simplifies the processes required for the rest of the router operation. The second method is Lowest Credit Weighted Fair Queuing or LCWFQ scheduler, that achieves a fairness index of 1 (One) and a work order of O(1) which is similar to the famous Deficit Round Robin (DRR) scheduling algorithm, except that the LCWFQ avoids the DRR weakness in implementation. The third method is an Adaptive Weight Adjustment (AWA) traffic shaper that the Internet Service Provider (ISP) may use when the total ISP commitments to the subscribers are more than what the network can handle. The AWA also serves the bursty streams when conflicting with the assigned SLA while maintains fair services to non-bursty traffic. The forth method is a three level memory index that fast search a record in the forwarding table where the number of memory accesses is always three no matter how large is the forwarding table. The thesis also includes two methods to translate the Service Level Agreement (SLA) into scheduling profiles and to report any bandwidth over-usage to help redefining the subscriber SLA. The proposed methods are designed to fit the network processors abilities to implement loosely coupled profiling, traffic shaping and scheduling.

Table of Contents

List of Tables

List of Figures

List of Acronyms and Symbols

ASD	Adaptive Scheduling Device
AWA	Adaptive Weight Adjustment
DC	Deficit Counter
DRR	Deficit Round Robin
ERR	Elastic Round Robin
GPS	Generalized Processor Sharing
IP	Internet Protocol
ISP	Internet Service Provider
LCWFQ	Lowest Credit Weighted Fair Queuing
LR	Latency Rate
LI	Lawful Intercept
MRR	Mini Round Robin
NP	Network Processor
PDRR	Pre-order Deficit Round Robin
PERR	Prioritized Elastic Round Robin
QoS	Quality of Service
RF	Relative Fairness
RFB	Relative Fairness Bound
RR	Round Robin
SC	Surplus Counter
SCFQ	Self-Clocked Fair Queuing

SFF	Smallest Finish-time First
SFQ	Start-time Fair Queuing
SLA	Service Level Agreement
SMM	Selective MultiMedia
SRR	Surplus Round Robin
SSF	Smallest Start-time First
TE	Traffic Engineering
UA	User Agent
VoIP	Voice over IP
WF2Q	Worst-case Fair Weighted Fair Queuing
WFQ	Weighted Fair Queuing
WRR	Weighted Round Robin

Acknowledgement

I would like to express my sincere gratitude and appreciation to my supervisor, Professor Hussein Alnuweiri for his encouragement and support at all levels. Dr. Hussein Alnuweiri has constantly guided me to remain focused on achieving my goal and exploring in depth and many thanks for his support in funding this research.

I would like to thank Dr. Steve Wilton and Dr. Rabab Kreidieh Ward for agreeing to be on my thesis committee and reviewing my thesis.

I would like to thank all the people who made this work possible. Thanks to Mollalign Assefa for providing expertise C++ and for the help in lab experiments. I would like also to thank Yasser P. Fallah and Tariq Al-Khasib for the critical comments and the helpful suggestions.

I really appreciate the wonderful Research Assistance program provided by The University of British Columbia. The program helped me to give up my previous job and get dedicated for my thesis. Thanks to UTStarcom for offering better job afterwards and funding the last part of the research. I would like to mention my mangers, Wael Menyawi and Jack Mar for their support in the last two years and Joe Perrella for the appreciation.

My gratefulness is to my wife Mosheera for the inspiration and the moral support. Raising four kids is challenging when living on one income most of the time.

Dedication

To Mosheera

1 Introduction

The Intel IXP2400 is an example of a network processor that consists of one Intel XScale core processor and eight micro-engines. The micro-engines are organized into two clusters of four, all on the same die. Applications can be implemented using the network processor to perform pipeline processing in which the different processors exchange packet descriptor information through Next-Neighbor (NN) registers or shared memory.

The thesis is proposing new methods to implement multimedia routers using the advantages of the network processors. The proposed methods are tested heuristic methods and focus on simplifying the implementation of the multimedia router. The methods are implemented in the proxy agent module, the scheduler module, the traffic shaper module, the forwarder three-level index module and the SLA manager module. In general, the Service Level Agreement (SLA) defines the profiles and the scheduler serves the streams according to their profiles. The streams that are classified under the same SLA profile are also expected to receive the same level of network services.

It is a complex task for the Internet Service Provider (ISP) to recalculate the bandwidth share per subscriber every time a new SLA is in action and added to the service. In some cases, the total commitment for the ISP to subscribers may be more than what the network can handle.

The thesis proposes methods to help design multimedia routers that are capable of reading the SLA parameters, recalculate the bandwidth per each SLA and schedule the traffic to provide fair bandwidth allocation for each SLA in action.

1.1 Contributions

The main contributions of this thesis are summarized hereinafter as follows:

(1) High level architecture design of selective multimedia (SMM) router to show where the proposed Scheduler, Profiler and Traffic Shaper fits

(2) Design and simulation of advanced complex scheduling algorithm referred hereinafter as the Lowest Credit Weighted Fair Queuing (LCWFQ)

(3) Design and simulation of adaptive weight adjustment traffic shaper

(4) Introducing methods to input the SLA parameters into the traffic shaper and recalculating the bandwidth allocated to each profile. Also introducing a method to provide feedback to the network manager when over commitment occurs

(5) Implementing a simulator and traffic generator that mimic network traffic and configuring traffic parameters like silent gaps, bursts and streams

1.1.1 Architecture Design of SMM Router

The Selective Multi-Media (SMM) router architecture includes two main contributions, a design of an SMM Proxy Agent and a Forwarder. The design of the proxy agent turns the router into multimedia aware router. The proxy agent selects the multimedia streams from the Egress modules and provides priority services to them. The proxy agent populates a proposed indexed forwarding/routing table for the active streams and adhere the profile tag to each record in the forward table. Moreover, it provides time stamp for each record for the purpose of aging and cleanup.

On the other hand, the proposed SMM forwarder performs forwarding functions that are controlled by the proxy agent. Strictly speaking, the SMM forwarder is unaware of the type of the stream it processes, while the proxy agent is the one that determines the type of the multimedia streams. The proxy agent makes the router capable of recognizing and handling different multimedia streams, by simply configuring the proxy agent.

The thesis also proposes a memory indexing method to improve the suitability of the network processors in implementing the multimedia routers. The index method limits the search for a route record in a forwarding table to three memory accesses. Regardless of how large is the forward table; the number of memory accesses is fixed. In a worst-case scenario, more than 100,000 records in a table can be searched in only three memory accesses by using less than 16MB of memory. A 16MB of memory is affordable internally in most of the network processors.

1.1.2 LCWFQ Scheduler

The Lowest Credit Weighted Fair Queuing (LCWFQ) scheduler is a new innovative algorithm and is considered as the main contribution for the thesis. LCWFQ algorithm has some similarity with the famous Deficit Round Robin (DRR) scheduler. The LCWFQ algorithm performs as good as the famous Deficit Round Robin (DRR) scheduling algorithm. LCWFQ is relatively easier to implement on NP, and avoids the DRR weaknesses and implementation problems.

1.1.3 Adaptive Weight Adjustment Traffic Shaper

The Adaptive Weight Adjustment (AWA) method is designed to control the bandwidth allocation according to the composition of the existing measured traffic, and the SLA assigned bandwidths to reshape traffic congestion. AWA is considered as an extended function to the scheduler where it feeds back the scheduler with the status of the traffic in the backlog, recalculates the bandwidth assigned to the profiles, and adjusts the priority to achieve the optimum fairness. The AWA is a traffic shaping technique and the word "Traffic Shaper" is used alternatively through the thesis for simplicity.

1.1.4 Other Contributions

The thesis also proposed a mechanism to translate the Service Level Agreement SLA into profiles and assure fairness to all subscribers. More contributions added to the thesis, which is implementing a simulator and traffic generator that mimic network traffic by configuring the parameters like the silent gaps, the bursts and the packets. The simulator provides more of a deep look into the backlog traffic in the scheduler input buffer.

1.2 Thesis Outline

This thesis introduces a multimedia aware router architecture that can be implemented on the IXP Network Processor (NP) platform. The thesis is organized as follows:

- Chapter 1 is this introductory chapter.
- Chapter 2 is an overview on the IXP NP platform.

- Chapter 3 describes the design of the proposed router architecture based on the IXP NP platform. In this chapter, the relevant NP modules and features, and the implementation of the respective network services are described in details.
- Chapter 4 describes three of the commonly known scheduling algorithms called the First-In-First-Out (FIFO), the Round Robin (RR) and the Deficit Round Robin (DRR) scheduling algorithms.
- Chapter 5 introduces a proposed scheduling algorithm named the Lowest Credit Weighted Fair Queuing (LCWFQ) and describes its operation. The chapter also describes a proposed adaptive weight adjustment method.
- Chapter 6 presents the simulation technique used to proof and justify the suitability of the proposed methods for the network processors based multimedia routers. The chapter demonstrates advantages of the LCWFQ proposed scheduling algorithm and presents simulation results of the performance of the scheduler and the adapted weight adjustment.
- Chapter 7 concludes the thesis and summarizes simulation results.
- Chapter 8 is the bibliography and references
- Appendix A describes the Class Description for the simulator
- Appendix B includes GUI screenshots

2 Network Processors

2.1 Introduction

Network Processors (NP) are optimized for packet processing and offer performance and flexibility for implementing network services. Network Processor also allow the addition of the new network services while maintaining high packet throughput and low packet latency. The NP meets the network performance and flexibility requirements through highly parallel, programmable architectures. The parallel nature of network processors allows processing of multiple packets simultaneously, which can greatly increase the throughput. This chapter provides a brief overview of the Intel IXP2400 Network processor that is relevant to the design of network processor based router design discussed in the next chapter [1].

The Intel IXP2400 Network processor consists of one Intel XScale core processor and eight micro-engines. The micro-engines are organized into two clusters of four, all on the same die. The Intel XScale core is an Advanced Reduced Instruction Set Computer (RISC) machine. The micro-engines are also RISC processors optimized for fast-path packet processing. IXP2400 has one scratchpad-hash-Control and status Access Proxy (CAP) unit, one Media Switch Fabric unit, one PCI controller, two SRAM controllers and one-DRAM controller.

The Intel XScale Core is compliant with the ARM V5TE architecture as defined by ARM Limited and has support for ARM's thumb instructions that allow the Intel XScale core to switch back and forth between the standard 32-bit instruction set and a 16-bit instruction set for better performance.

2.2 Micro-engines

Each micro-engine has an independent instruction store large enough for 4K of 40-bit instructions that is initialized by a code in the XScale core. The micro-engine has eight hardware-based threads of execution, which can be configured to use either eight or four threads. The threads are non-preemptive, which means that the currently active thread must explicitly release control of the processor before another thread can run. This nature of the threads simplifies synchronization within the micro-engine.

2.2.1 Registers

The micro-engines have four types of registers: general purpose, SRAM transfer, DRAM transfer, and Next Neighbor (NN). Micro-engine registers do not need to be flushed to memory when the control of the micro-engine switches from one thread to another because the hardware allocates an equal portion of the total register set to each micro-engine thread.

Each micro-engine has 256, 32-bit General Purpose Registers (GPRs), allocated into two banks of 128 registers. The GPRs per-micro-engine can be accessed in thread-local or absolute mode. In thread local mode, each thread accesses a unique set of GPRs. If the micro-engine configured to execute eight threads, 32 GPRs are allocated to each thread. However, if is configured to execute four threads, 64 GPRs are allocated to each thread.

Each micro-engine has 256, 32-bit SRAM transfer registers that are used to read from and write to all functional units on the Intel IXP2400 network processor except for the DRAM unit. SRAM transfer registers are the primary mechanism for dealing with asynchronous memory operations.

Each micro-engine has 256, 32-bit DRAM transfer registers divided equally into read-only and write only. DRAM transfer registers are used for communication between the micro-engines and the DRAM unit can be used for read-only communications with the other hardware unit. Like the SRAM transfer registers, DRAM registers can be accessed in thread-local and global manners.

Each micro-engine has 128 32-bit next neighbor registers. These registers can be used in one of two modes. The first mode makes data written in these registers available in the next micro-engine. Assisted by two Control and Status Registers (CSRs) in the micro-engine, the next neighbor registers are used as a 128-entry First-In-First-Out (FIFO) queue. In the second mode, the registers are used as extra general purpose registers. Each micro-engine has 640 long words of local memory. Any thread in the micro-engine can access data in this memory with at most three-clock-cycle latency, much faster than the scratchpad, SRAM or DRAM memory.

2.3 Memory

The IXP2400 Processor can access three external memory types the scratchpad, the SRAM and the SDRAM. All micro-engines as well as the Intel XScale Core processor share these memory interfaces.

Scratchpad memory is a 16K on-chip memory organized as 4K 32-bit words. It provides a small low-latency interface to all of the micro-engines. It supports operations that are ideal for keeping counters across multiple micro-engine threads as well as synchronizing access to data structures across micro-engines.

Unlike scratch pad memory, SRAM is off-chip. The IXP2400 processor only provides an interface to the actual SRAM. This interface is embodies in the micro-engine's SRAM unit. Each SRAM unit provides an interface for up to 64 Mbytes of memory. In addition, each SRAM unit contains a 64-element queue array. Like the SRAM memory, SDRAM memory is external to the IXP2400 processor. The DRAM unit provides an interface for up to one Gigabyte of high throughput memory.

3 Network Processor Based Multimedia Router

3.1 Introduction

This chapter presents an architecture design that exploits the implementation of pipelined router architecture. A simple diagram of the proposed router architecture is shown in Figure 3.1 where it shows User Agents (UA) where each UA represents a multimedia client/server that may establish connection to another multimedia UA server/client. The router serves also non-multimedia Terminals (T) connected that may send, and receive non-multimedia streams like FTP, HTTP and text messaging. The first stage shown in the figure is a simple packet classifier in the ingress module that sends all received packets to an SMM forwarder while it picks the packets that should be distant to SMM proxy and sends a copy to a proxy agent.

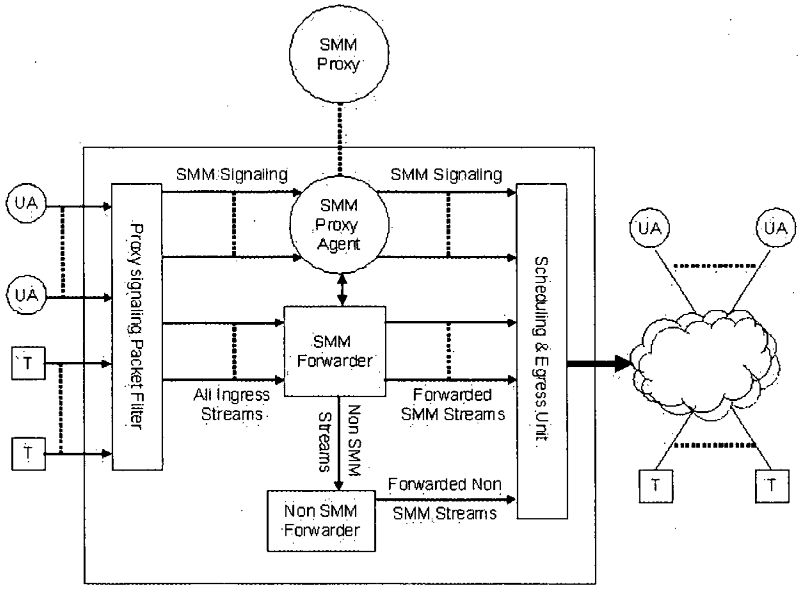


Figure 3.1: Simple Diagram of the proposed router architecture

The purpose of the Ingress Packet Classifier is to determine the packets distant to a proxy server configured in the proxy agent, while the purpose of a proxy agent is to turn the router into multimedia aware router. The proxy agent can be programmed to serve selected multimedia streams that pass through the router and known to the agent. The router contains a Selective MultiMedia (SMM) forwarder that performs forwarding to the multimedia streams selected by the proxy agent. Strictly speaking, the SMM forwarder is unaware of the type of the stream it processes while the proxy agent is the one that determines the type of the streams, and builds the SMM forwarder lookup table.

To meet the quality of service requirements for multimedia networks and to enhance the search speed for the lookup tables in routers, the thesis proposes a fast method for searching the lookup table. The method is an indexing scheme that makes the search time for a record always constant regardless of the number of records in the lockup table. The lockup table record contains profiling information and forwarding information that is obtained from a static database and is updated by the network administrator. The forwarding information is obtained from the conventional routing table of the router. When a new stream starts, the proxy agent is the one that updates the stream record in the SMM lookup table during the stream connection establishment.

Figure 3.2 shows the packet classification and forwarding where the Selective MultiMedia (SMM) classifier classifies the packets it receives into packets distant to a proxy servers and non-proxy packets that are not distant to a proxy server. The packet distant to the proxy server passes through the proxy agent. Then, the proxy agent modifies the lookup table of the SMM forwarder. The forwarder knows from the updated lookup table which streams are Selected Multimedia streams and push the stream forward to the scheduler, while the non-selected multimedia streams passes through the normal routing and forwarding module.

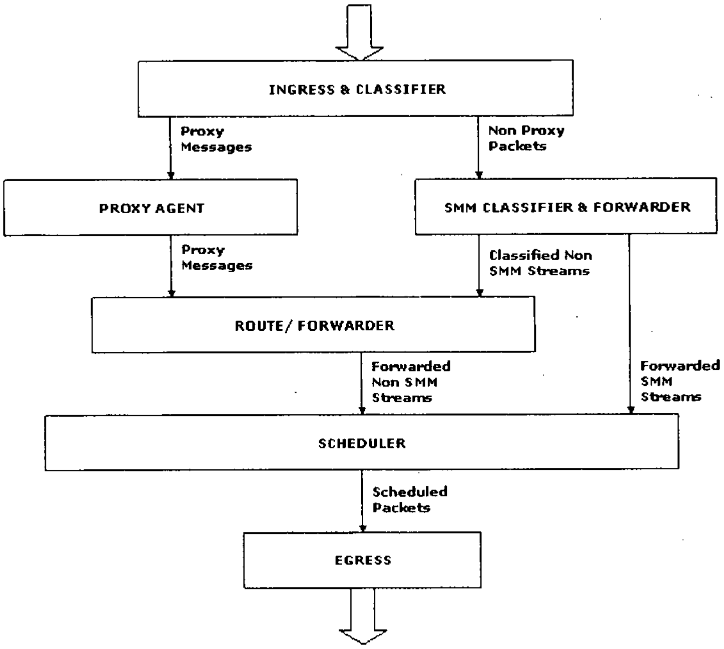


Figure 3.2: Packet Classifying & Forwarding Diagram

Next, the streams are scheduled according to their profiles assigned by the scheduler that uses a new proposed scheduling algorithm known as the Lowest Credit Weighted Fair Queuing (LCWFQ) and the details of the algorithm are discussed in Chapter 5. the last module in the router is the egress module that transmits the packets to the external network.

3.2 Proxy Agent - Proxy Server Relationship

The User Agent (UA) sends signaling messages to the proxy Server to establish a connection to a remote UA. As explained earlier the router has a non-multimedia forwarding tables and a multimedia forwarding table. The UA signaling message uses the non-multimedia routing and forwarding module, as it does not need to have a high priority. Afterwards, the proxy server contacts the distant User Agent (UA) to establish the connection. In doing so, the normal routing module finds the routing information to the distant UA. Simultaneously, the proxy agent reads a copy of signaling message, which is distant to the proxy server that is known to the agent [2].

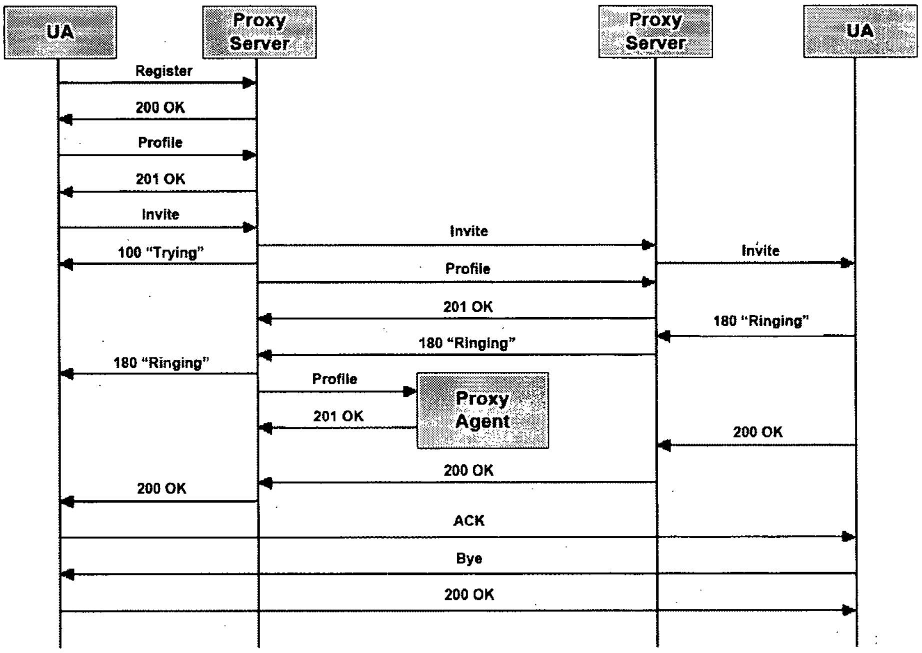


Figure 3.3: Proxy Agent – Proxy Server Relationship Diagram

3.3 Architecture

As shown in Figure 3.4 below, the proposed router has six modules, which are the Ingress, the Proxy Agent, the SMM Forwarder, the Route Forwarder, the Scheduler, and the Egress. The proposed router has a pipelined architecture taking the advantage of the of the network processor where each module is implemented in its own micro-engine and executed independent of each other (loosely coupled)

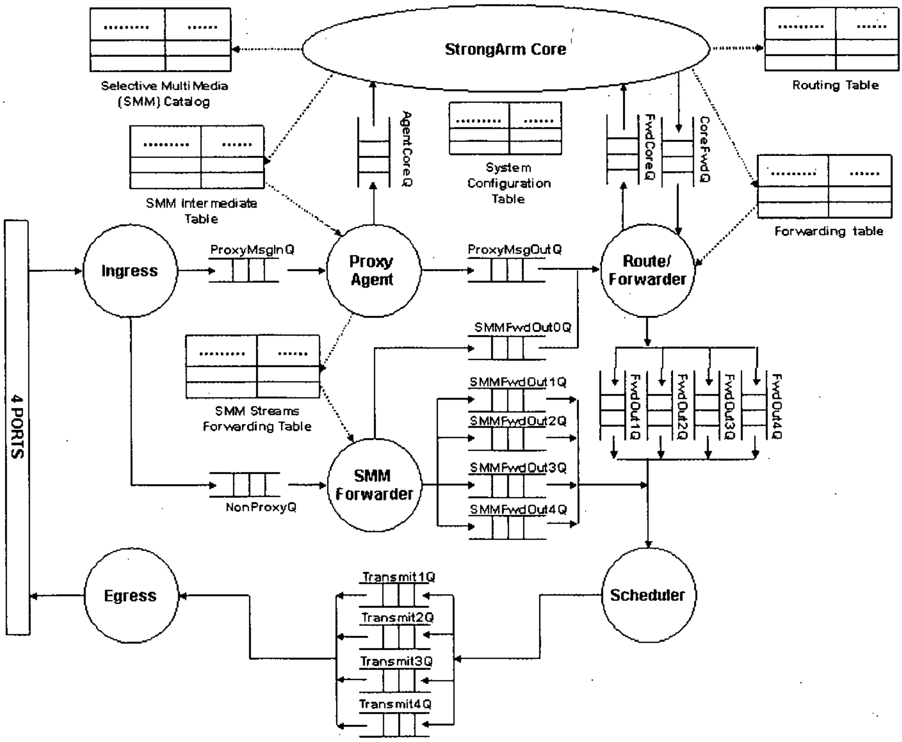


Figure 3.4: Detailed architecture of the SMM Router

3.4 The Ingress

The Ingress unit receives packets at high speeds and maintains two queues, one queue to communicate with the Proxy Agent, which is known as the proxy queue, and the second is the non-proxy queue that communicates with the SMM Forwarder. The Ingress module maintains the IP address of the Proxy Server represented by the proxy agent.

3.5 The Proxy Agent

When a User Agent (UA) seeks to communicate with another UA, it undergoes a sequence of steps that enables the UA to establish a communication session. The Proxy Agent handles these steps. Upon successful reception of a packet, the Ingress module retrieves the destination IP address from the DRAM where the packet is stored. It then compares the destination IP address with the IP address of the Proxy Server. If it matches, the packet is sent to the proxy queue or else to the non-proxy queue.

Assuming that the proposed router has eight ports, the Ingress will have eight hardware-assisted threads at its disposal. Each thread is associated with one port and performs similar operations on the packets received from that port. The Ingress unit design is modified to include a first level classifier to identify the multimedia signaling data with a destination address pointing to a proxy server previously configured in the proxy agent. It is assumed that the implementation of the router provides a way to set the address in the multimedia proxy server.

Part of the responsibility of the proxy agent is to identify the content of the packet to determine the proxy server of the UA. The Proxy Agent needs to know the address of each proxy server and can be as many proxy servers as needed. The addresses of the proxy servers are maintained in the *proxy agent* module. All packets sent to the proxy are kept in a single queue called *ProxyQ* queue. All other packets are sent to a single FIFO queue called *NonProxyQ*. The Proxy Agent partially depends on the StrongArm XScale Core, hereinafter referred to as the Core.

Upon retrieving packets from *ProxyMsgInQ*, the first operation performed by the Proxy Agent is to copy the packet pointer into the output queue called *ProxyMsgOutQ*. This method enables the next module, which resides in a different micro-engine, to carry on processing the packet.

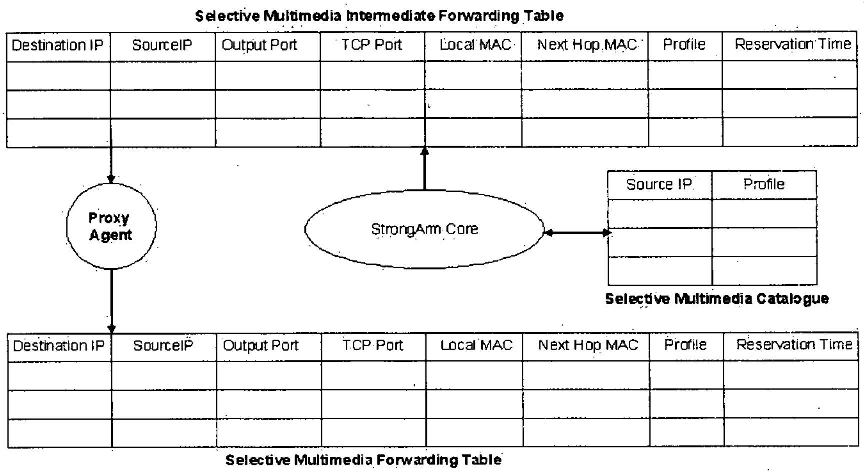


Figure 3.5: Operational relationship between the Core and the Proxy Agent

Figure 3.5 above shows a Proxy Agent makes use of three lookup tables called SMM Catalogue, SMM Forwarding Intermediate Table and SMM Forwarding Table. The SMM Catalogue is a static data table that associates the source address of the SMM UA to its assigned profile. The catalogue is initialized and then updated through the Core processor. Multiple User Agents may be assigned identical profiles. The network administrator assigns the profiles to indicate the service level for each UA. SMM Forwarding table is a data structure stored in the SRAM that is used to store records of composed information retrieved from the SMM Catalogue and the routing table.

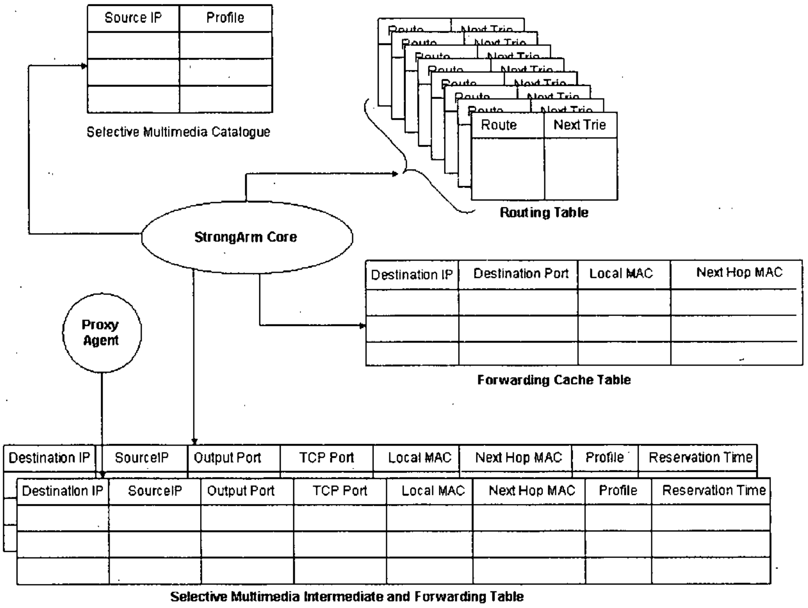


Figure 3.6: Relationship between Forwarding and routing tables

Unlike the Ingress module, the Proxy Agent needs to know the signaling protocol and has to be able to identify when a sender UA invites another UA for a communication session and when the receiver UA accepts the invitation. When the Proxy Agent detects an *INVITE* Message, the agent communicates with the Core processor so that the core loads all the necessary routing information into the SMM Forwarding Intermediate table. This is because an *INVITE* message most likely will be followed by an *OK* message and the communication begins. Retrieving the routing information before the communication begins will speed up the process. When an *OK* message is received, the stream record is marked active, and then is used later to assemble the packet to forward it to its destination. In fact, the SMM Forwarding Intermediate table and the SMM Forwarding Table are implemented as a single table with the addition of a flag field in the data structure. Thus, the two tables are implemented as contiguous array of records interleaved with each other. The records marked active are those records for which the *INVITE* and *OK* messages have been received and the records marked inactive are those records for which only the *INVITE* message has been received [3] [4].

3.6 The SMM Forwarder

SMM Forwarder is a module running in a single micro-engine that is responsible for classifying incoming packets into multimedia and non-multimedia packets. It places the classified packet record into a queue identified by the port number while the non-multimedia packets are placed in a single queue, which is a shared memory by the SMM Forwarder and the Route/Forward unit.

The SMM forwarder communicates with the Ingress module through a FIFO queue implemented in the SRAM. Each queue element contains a pointer to a DRAM unit where the packet under consideration is stored. The proposed SMM forwarder and profiler modules create a common data structure that contains the pointer to the packet, the source and destination IP addresses, and the length of the packet. It uses the source IP address to index the active catalogue maintained in the SRAM from which the profile of the source are retrieved. The profile ID and the relevant data are then added to the data structure created. These data are then placed in a different SRAM queue that serves as communication channel between the SMM Forwarder and the Scheduler. If a route record corresponding to a source is not found in the SMM forward table, the source record data is passed to the normal Router/Forwarder module [5].

The SMM Forwarder identifies whether the packet is a multimedia or not by looking up the SMM Forwarding table updated by the Proxy Agent. The packet is identified as multimedia packet if the source IP address of a packet is found in the SMM Forwarding table. Then the forwarder passes the profile information to the scheduler. If the packet is non-multimedia, then the SMM forwarder send it to the normal forwarder to find the routing information in conventional way. Non-multimedia packet gets the lowest profile then sent to the scheduler.

3.7 The Forwarder

The Route/Forward module is responsible for routing and forwarding all non-multimedia in addition to the multimedia signaling packets. It initializes and maintains the routing table to determine the output port path of the packets. After determining the route for a packet, both the Proxy Agent module and the Route/Forward module transfer the records for the packets under consideration to the respective queues. The scheduler dispatches the packets to the Egress module, which performs the actual transmission of the packets to the external network. The next chapter provides more details on the scheduler.

Because of the real time requirements of multimedia streams, it is important that the search of a record corresponding to the packet under consideration is time efficient. For this purpose, the thesis introduces a three level indexing scheme discussed next.

3.7.1 Three-Levels Indexing

In the implementation of the proposed router, the lookup tables are designed to keep the routing information in the SDRAM. However, retrieving the information with fewer memory accesses is critical. The thesis proposes Three-Levels Indexing method for route storage scheme whereby the route of any destination IP address is determined by three memory accesses as illustrated in Figure 3.7.

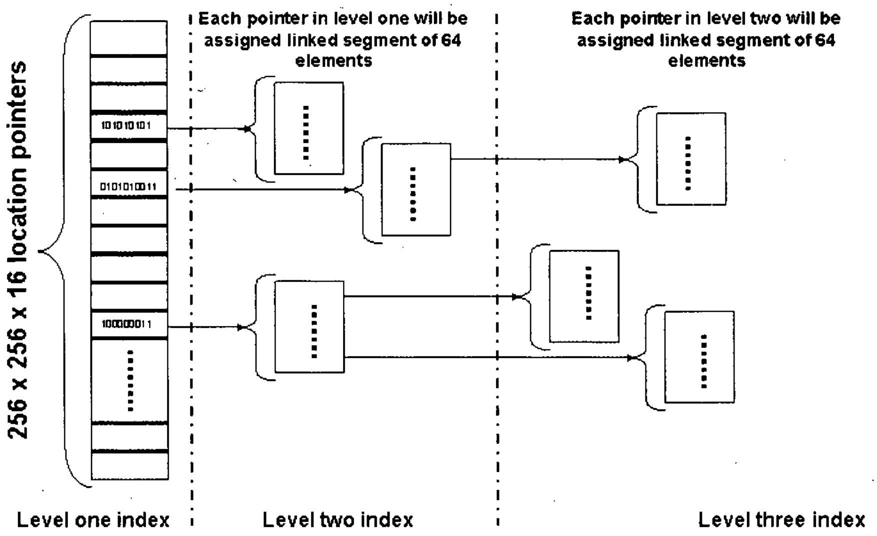

256 x 256 x 16 location pointers
1010101101
0101010011
100000011
Each pointer in level one will be assigned linked segment of 64 elements
Each pointer in level two will be assigned linked segment of 64 elements
Level one index
Level two index
Level three index

Figure 3.7: Three Levels Indexing

The first level is an array of one million SRAM pointers pointing to a second level of memory segments, which uses the most significant 20 bits of the destination IP address. The second level, indexed by the next 6 bits of the IP address, points to a memory segment storing 64-DRAM memory locations represented by the last 6 bits. The third level of the memory segment is stored in the DRAM and contains the addresses where the stream information is stored.

This mechanism insures that the time required finding a record is always 3 memory accesses regardless of the number of records. The lookup table may hold hundreds of thousands of records or even millions while keeping the search time fixed. The record may include the destination port, the multimedia profile tag, the pointer to the actual packet, and any other information that may be required. The record is used in the forwarding process, scheduling process, and routing process.

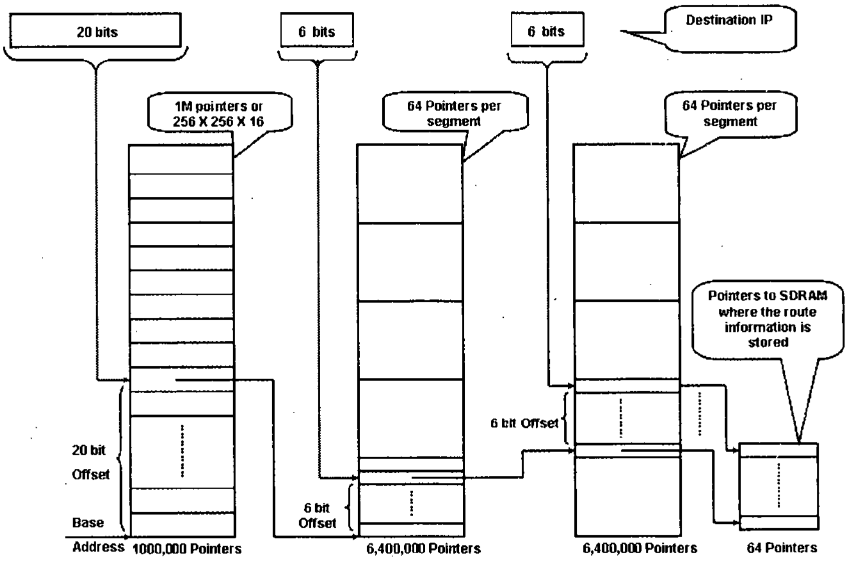


Figure 3.8: Forwarding Table Structure

In an example of a worst case scenario, the maximum number of memory locations to build the three levels index is equal to the first level 20 bits (1,000,000) x second level 6 bits (64) x third level 6 bits (64) = 4 Giga locations, which practically will not happen. Typically, for 64,000 streams routes, it needs 100,000 locations out of the first 20 bits index and that reduces the total number of locations to the following:

L1 (1,000,000) + L2 (100,000 x 64) + L3 (100,000 x 64) = 13,800,000 locations.

Similarly, one million records need 129,000,000 index locations, which is still affordable by most of the available network processors. The three-levels indexing method is scalable to accommodate larger forwarding tables. In this way, one forwarding table may hold millions of records and searching time stays fixed unlike the hashing searching method and the binary searching method where the scalability costs more delay. However, if the memory cost is a concern, then more levels of indexing may be considered [6][7][8].

3.8 The Scheduler

There is a complexity in implementing the per-flow scheduler in the most commonly used core routers. Routers use aggregate scheduling rather than per-flow scheduling. The proposed scheduler module provides a simple method to regroup the individual streams into aggregate flows and turn the per-flow scheduling to aggregate scheduling. The scheduler uses the time waiting the Egress to transmit one packet to turn on Adaptive Weight Adjustment (AWA) scheme. The AWA calculates the backlog traffic in the scheduler input queue and readjust the aggregate profiles accordingly. For simplicity, the AWA scheme will be referred through the thesis as the traffic shaper. The traffic shaper looks at the backlog packets and adjusts the aggregate profile weight. The scheduler uses the calculated aggregate profiles.

3.9 The Egress Module

The Egress module transmits the scheduled packets to the external network. The Egress is deployed on one micro-engine. Each port requires a dedicated micro-engine thread. When it dispatches packets, the scheduler uses as many queues as the number of ports.

4 Selection of a Suitable Scheduler

4.1 Introduction

In this chapter, there is a consideration of three scheduling algorithms called First-In-First-Out (FIFO), Round Robin (RR), and Deficit Round Robin (DRR). The chapter discusses the merits of these algorithms to determine their suitability in the real time system. The next chapter will introduce the Lowest Credit Weighted Fair Queuing that has a predictable behavior and effective packet processing cost.

4.2 First-In-First-Out (FIFO)

In FIFO scheme, the order of the packet arrival determines the allocation of packets to output buffers. This algorithm assumes that congestion control is performed by the source itself. To achieve this purpose it uses a feedback scheme. In feedback schemes for congestion control, connections are supposed to reduce their sending rate when they sense congestion. However, a rogue flow can keep increasing its share of the bandwidth and cause other well-behaved flows to reduce their share. If a rogue connection sends packets at a high rate, it can capture an arbitrary fraction of the outgoing bandwidth.

4.3 Simple Round Robin

Simple Round Robin is a scheme by which every profiled queue is serviced at least once in every round. Ordinary round robin servicing of queues can be done in constant time. The major problem, however, is the unfairness caused by possibly different packet sizes used by different profiles. For example, if, hypothetically, the low priority profile keeps sending long packets while the high priority profiles have short packets, then the low priority profile monopolizes the use of the bandwidth. Therefore, this scheme is not suitable for scheduling multimedia streams [9] [10].

4.4 Deficit Round Robin

Deficit Round Robin is a scheduling algorithm [11][12] whereby the scheduler transmits packets from different profiles based on predetermined quantum values assigned to each of these profiles. The scheduler allows each profile to send packets subject to the length of the packet. The packet length, expressed in bytes, is less than or equal to the sum of the quantum value assigned to the profile and the value contained in a state variable called the Deficit Counter, which contains the difference between the assigned quantum value and the number of bytes transmitted in the previous round. If there are no more packets in the queue, the scheduler resets Deficit Counter to zero. Otherwise, it holds the difference between the assigned quantum value and the length of the packet sent.

DRR provides an isolation mechanism that achieves nearly perfect fairness, and also takes O (1) work to process a packet. The scheme is simple to implement at a router or a gateway. The main difference between DRR and RR is that if a queue was not able to send packet in the previous round because its packet size was too large, the remainder from the previous quantum is added to the quantum for the next round. Thus, deficits are tracked and recorded, and hence, queues that were shortchanged in a round are compensated in the next round.

4.4.1 DRR Weakness

The empty queue introduces errors for the DRR process. To avoid empty queues, the algorithm keeps an auxiliary list called the Active List, which is a list of indices of queues that contain at least one packet. The packet is added to the end of Active List when it arrives to a previously empty queue. Whenever the index is at the head of the Active List, the algorithm services the corresponding queue by scheduling packets whose combined length is less than or equal to the sum of the Quantum value and the Deficit Counter. If at the end of this service opportunity the queue still has packets to send, the index is moved to the end of Active List, otherwise the Deficit counter is set to zero.

The size of the various quantum variables in the DRR algorithm determines the number of packets that can be served from a queue in a round. This means that the latency for a packet and the throughput of the router are dependent upon the value of the quantum variable. The thesis introduces a new scheduling algorithm to avoid this dependency. The new scheduling algorithm called the Lowest Credit Fair Queuing that has work load of O(1) and achieves a fairness index of 1 (one), which is discussed in the next chapter.

4.5 The SFQ and SCFQ

These schedulers are time-based schedulers [13]. The LCWFQ, DRR, RR, and FIFO are frame-based schedulers [14][15]. However, the proposed adaptive weight adjustment method behaves when used with the scheduler it does fall in a category between the frame scheduling and time scheduling as it depends on the time spent by the Egress to transmit the last packet to recalculate and redistribute the bandwidth shares.

5 Scheduler and Traffic Shaper

5.1 Introduction

This chapter presents a proposed scheduling method as part of a router and called Lowest Credit Weighted Fair Queuing (LCWFQ). There is an additional module added to the scheduler, which is Service Level Agreement Manager module that allocates bandwidth to clients based on a specified Service Level Agreement (SLA).

The chapter presents also the proposed Adaptive Weight Adjustment method that may be used in some situations when the Internet Service Provider may be over committed to the subscribers. Another situation would be when there is a need to serve applications that normally are inactive unless required like the Lawful Intercept (LI) function. The LI function is intercepting with certain streams, copy the streams, and send them to a LI agent. The intercepted streams require a very small bandwidth, yet have a very high priority.

5.1.1 Per-Flow scheduling Vs Aggregate Scheduling

The thesis proposes the LCWFQ, which is a weighted fair queuing scheme. LCWFQ is used to schedule flow traffic in a per-flow scheduling mode. However, the LCWFQ is designed also to schedule the aggregate flows with the help of the proposed add-on modules like the SLA manager and the proposed AWA traffic shaper.

In order to use the LCWFQ with the aggregate flows, each flow must be assigned to a profile group and then each group is assigned to an aggregate flow. Each aggregate flow must have a new profile value for the scheduler. The traffic shaper calculates these values as explained later. All flows belong to the same aggregate flow must have the same profile weight. The aggregate flow weight is calculated as the total number of streams/flows within the aggregate multiplied by the per flow weight originally assigned by the SLA. The thesis presents two schemes for the traffic shaper where one scheme is based on fixed weight assigned by the SLA while the other scheme is an adaptive weight adjustment calculated based on the backlog traffic in the scheduler input queue.

5.1.2 Forwarder, Profiler and Scheduler relationship

Since packet sizes are generally large to move internally between the internal modules, the router saves the packets in memory and work only with packet pointers. The payload data portion of the packet is not needed until the packet is ready to leave the router. Following this logic, the forwarding module passes one record for each packet, which contains the packet pointer, the packet length, and the packet profile. The profile of the packet represents the percentage quota of the bandwidth given to a packet stream from a specific source address. In the design of the high performance network processor based router, the Ingress module extracts information from the received packet and passes it through to the forwarder. Then, the forwarder calculates some other information to determine the route for each incoming packet and send it to the scheduler.

The profiler is a module that adds the profile information into a proposed forwarding table. It should be mentioned that the profiler reuses the forwarder information to classify the packets. The profiler in this case is running with a minimal processing cost since it reuses the existing information. Then the profiler passes the results into the scheduler input queue along with the packet descriptor information. The scheduler works with the classified packets that are placed in their own separate queues by the profiler. Figure 5.1 illustrates the profiling mechanism applied on three profiles, when the scheduler implements a round robin scheduling algorithm.

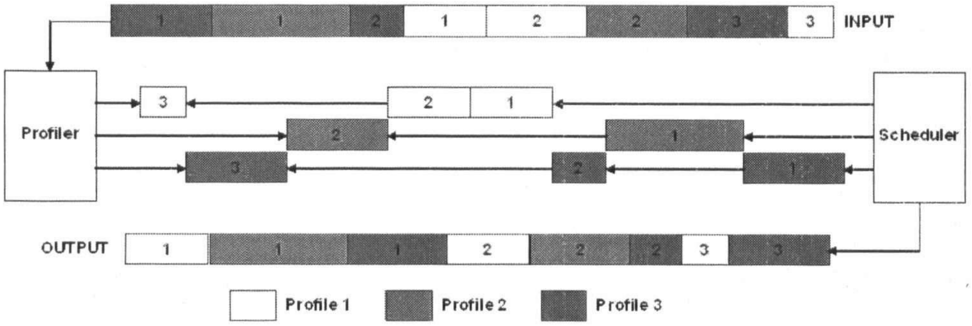


Figure 5.1: An illustration of the profiling mechanism for three profiles

5.1.2.1 Profiler Processing Cost

In order to depict the profiler processing cost, a sample calculation is provided below to calculate the total time required to build the profiler information for one packet in the worst condition when the packet is short, or in other words, 64 bytes long. A single access to the SRAM takes approximately 4 clock cycles and the maximum clock frequency of the IXP2400 network processor is 600MHz.

The proposed Three-Level indexing needs three memory accesses to read a record then one-memory access to write. Thus, the access time, as discussed in chapter 2, takes a total memory access time of

$$t_{read} = \frac{3*2\ \textit{Clock Cycles}}{600MHz} = 0.01\mu s$$

$$t_{write} = \frac{1*4\ \textit{Clock Cycles}}{600MHz} = 0.006\mu s$$

However, sending the same packet (64 bytes) over a line with a rate of 100Mbps line takes a transmission time, tx, of

$$t_x = \frac{8\frac{bits}{byte}*64bytes}{100Mbps} = 5.12\mu s.$$

Therefore, the maximum overhead delay, $t_{overhead}$, is approximately

$$t_{overhead} = \frac{0.016\mu s}{5.12\mu s}*100 = 0.003\%$$

This delay is considered negligible compared to delay caused by any queuing situation.

One of the conclusions is that by adding processing functions on early stages of the router, it saves processing time later. These proposed functions are the profiler, the Proxy Agent, the SLA manager, and weight adjustment traffic shaper. The proposed functions are simple and low cost processes. Moreover, these functions make the other modules like the scheduler and the forwarder processes simpler and shorter.

5.2 The Profiler

The purpose of the Profiler is to assign a profile tag to the incoming packet record and place the packet record in a queue designated for each profile. The profiler packet record is mainly the packet length, source IP and destination IP addresses and the profile tag. Normally, the profiler search for the profile tag for each incoming packets in the profiling table based on the source address. It is proposed that the profiler with the use of the SLA manager extracts the profile information from each SLA and populate the profiling table. It is proposed also that the profiler with the help of the weight adjustment, construct the aggregate flows and assign them new profile values.

In ASD Architecture diagram shown in figure 5.2, the profiler does calculations that describe the aggregate flow profiles and sends the results along with the packet headers to the scheduler input queue.

The profiler module classifies incoming packets according to their source information like the IP address, port number, and then obtains the profile information from the SLA manager. That makes the scheduler dedicated to perform packet scheduling based on the assignments prepared by the profiler. The record for each packet in the scheduler input queue contains:

1. The length of the packet
2. The profile identification
3. The total length of all previously arrived packets

4. The total length of all previously arrived packets from the same profile

5. The pointer to the next packet in the same profile

6. The pointer to the packet data

The Scheduler creates one queue for each physical egress port. These queues are shared memory regions, or Next-Neighbor registers in the IXP2400, used by the Scheduler and the Egress module to exchange packet records.

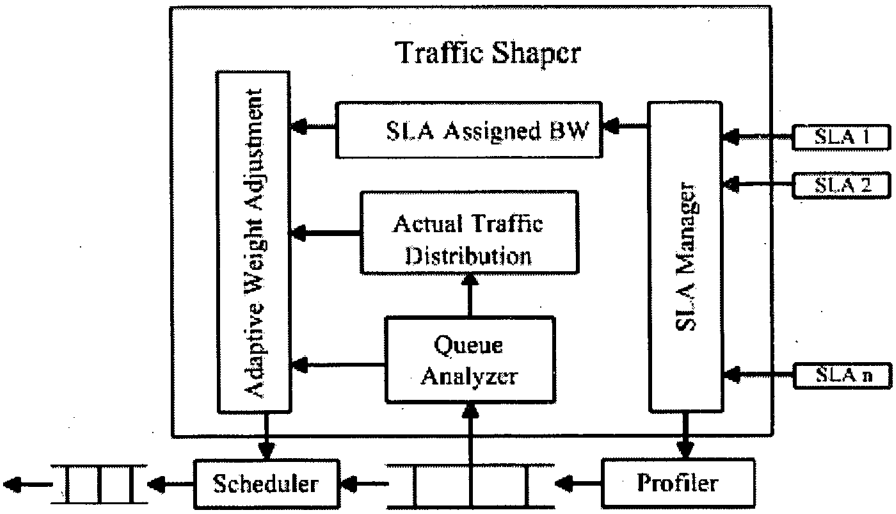


Figure 5.2: ASD Architecture

The Profiler is responsible for assigning profiles for incoming packets according to a profiling database maintained within the profiler application. Having done the profiling, the Profiler sets up a separate queue for each profile and places every packet in its own respective queue. In the proposed Multimedia Router application, the profiling database is a static data table based on the Service Level Agreements (SLA) and can be changed by the network administrator. The profiler is accessible to both the Traffic Shaper and the Scheduler.

5.3 Adaptive Weight Adjustment

5.3.1 Formulas and mechanism

Routers use aggregate scheduling rather than flow scheduling. The thesis proposes an Adaptive Weight Adjustment (AWA) heuristic method to provide a simple way to adjust the scheduling parameters and adapt the scheduler to serve aggregate flows. For simplicity, the adaptive weight adjustment will be expressed as the traffic shaper though the thesis. To explain the traffic shaper method, assume that the service provider signs hundreds of thousand of service level agreements with the subscribers. It is expected that the subscribers, over time, will generate distinctive flows equal to or more than the number of the signed agreements. However, these distinctive flows may be grouped into few aggregate flows based on their assigned profiles.

Similarly, the proposed traffic shaper reads the SLA parameters for each subscriber and catalogues the agreements into complete and detailed database of flow profiles. Then the traffic shaper works on the created database to link each flow profile to an aggregate flow profile. All flow profiles that belong to the same aggregate profile have the same type like VoIP, HTTP, FTP and also have the same characteristics like packet length and maximum accepted delay. This way, the number of aggregate flows will be few compared to the distinctive flows. Within the aggregate flow, all packets are treated similarly. The proposed traffic shaper puts each aggregate flow in a separate input queue to the scheduler. This way, it simplifies the scheduling process and the scheduler deals with aggregate flow as if it is a single flow.

During initialization, a proposed SLA manager module calculates the weight assigned to each aggregate flow and assigns it as initial value. Then after initialization, the adaptive weight adjustment or the traffic shaper starts. The traffic shaper measures the actual active aggregate flows, recalculates the aggregate flow weight, and then sends the new values to the scheduler. From figure 5.2 shown earlier, a queue analyzer unit determines the actual traffic distribution for all profiles. The traffic shaper receives the assigned bandwidth from the SLA manager and the actual traffic distribution from the analyzer. Then the shaper uses the assigned bandwidths and the actual traffic composition to determine the final weights for each aggregate profile, which is named within the thesis as the adaptive weight adjustment. The actual aggregate profile rate, $R_{act,p}$ is recalculated according to the following formula:

$$R_{act,P} = \frac{l_p}{\sum_{p=1}^{N_p} l_p},$$

Where l_p is the combined length of all the packets in aggregate profile p and N_p is the number of aggregate profiles.

If, $\forall p, R_{act,p} \leq R_{as,p} \Rightarrow R_{f,p} = R_{as,p}$, where p is the active aggregate profile number, and $R_{f,p}$ is the final calculated percentage of the active aggregate profile. Otherwise,

$$R_{f,p} = \frac{R_{act,p} + \min(R_{as.p}, R_{act,p})}{\sum_{p=1}^{N_p} (R_{act,p} + \min(R_{as,p}, R_{act,p}))}$$

Where $R_{as,p}$ is the assigned percentage for profile *p*. It is assumed that the above formulas expressed for a single output port. For any port ***j*** and profile ***i*** the two schemes are expressed as,

$$R_{act,Port[j]}[i] = \frac{profileLength[i]_{Port[j]}}{\sum_{i=0}^{n-1} profileLength[i]_{port[j]}}$$

And

$$R_{adapted,port[j]}[i] = \frac{R_{actual,port[j]}[i] + \min(R_{assigned,port[j]}[i], R_{actual,port[j]}[i])}{\sum_{i=0}^{n-1} (R_{actual,port[j]}[i] + \min(R_{assigned,port[j]}[i], R_{actual,port[j]}[i]))}$$

Table 5.2 and figure 5.4 are numerical examples for the above formulas and demonstrate the effectiveness of the traffic shaper, which is based on adapted weight adjustment. For example, the actual traffic of profile 1 is greater than the assigned value while profile 2 and profile 3 have a traffic level less than the assigned. In this case, the traffic shaper distributes the BW such that active aggregate profile 1 gets more than the assigned BW while profile 2 and 3 are assigned a value of BW that is just enough to accommodate their needs. In addition, even though profile 4 has the lowest priority (as determined by the initial BW assignment), it is allocated more than its assigned value.

	P1	P2	P3	P4
Assigned BW (%)	40	30	20	10
Actual Traffic (%)	50	10	10	30
Minimum of Actual and assigned R_{min}	40	10	10	10
Sum of Actual and R_{min}	90	20	20	40
Adapted BW Assignment	52.94	11.76	11.76	23.53

Table 5.1: Illustration of Traffic shaping in the case of four profiles

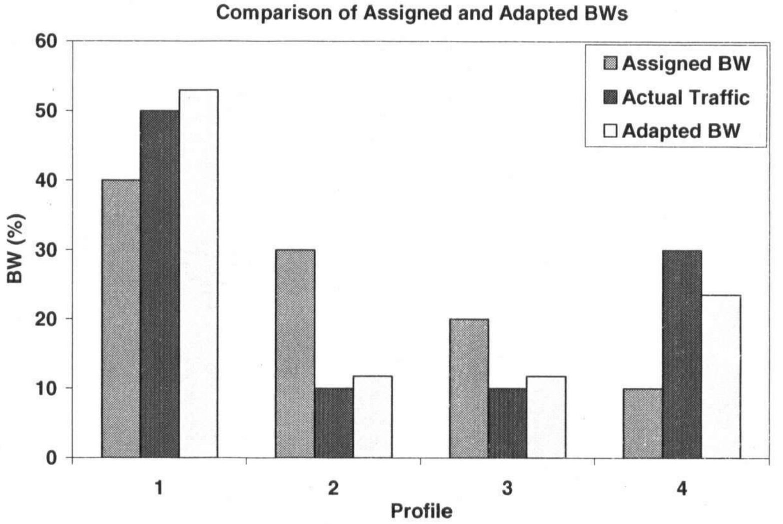


Figure 5.3: BW adjustment performed by the Traffic Shaper in case of four profiles

The traffic shaper used the unallocated bandwidth from the aggregate profiles that are not using all of their assigned bandwidth and then reassigns the unused bandwidth to active aggregate profiles that needs it. The traffic shaper may also be configured to set threshold alarms when a certain aggregate profile exceeds its usage of the allocated bandwidth. Table 5.3 and figures 5.4 show another example using nine profiles.

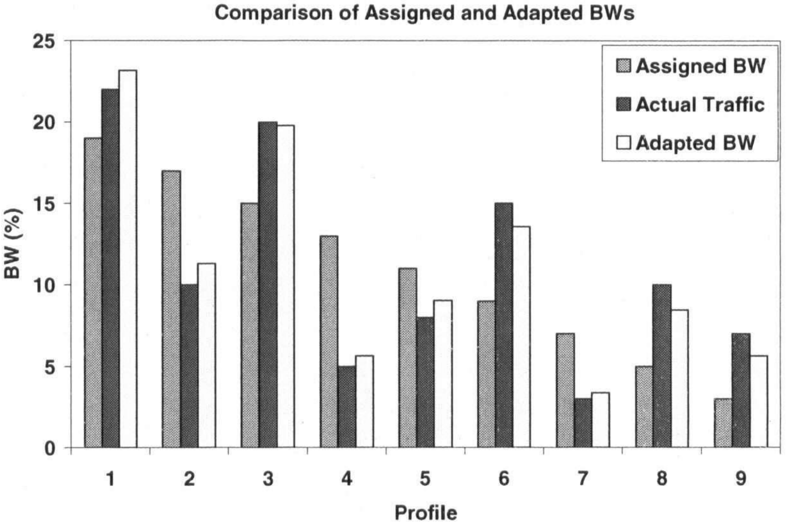


Figure 5.4: BW adjustment performed by the Traffic Shaper in case of nine profiles

	P1	P2	P3	P4	P5	P6	P7	P8	P9
Assigned BW (%)	19	17	15	13	11	9	7	5	3
Actual Traffic (%)	22	10	20	5	8	15	3	10	7
Minimum of Actual and assigned R_{min}	19	10	15	5	8	9	3	5	3
Sum of Actual and R_{min}	41	20	35	10	16	24	6	15	10
Adapted BW Assignment	23.2	11.3	19.8	5.7	9.0	13.6	3.4	8.5	5.7

Table 5.2: Illustration of Traffic shaping in the case of nine profiles

In the proposed network processor architecture, the Traffic Shaper and the Scheduler reside on the same micro engine. Therefore, the adapted percentage, which is calculated by the Traffic Shaper, is communicated to the Scheduler through shared registers, called absolute registers in the IXP2400 NP. If the number of profiles is too large to be contained within the registers, it is spilled to the local memory of the micro engine. The local memory has a lower latency than all the three different kinds of memory the IXP2400 supports, and hence the performance penalty has low significance.

In relation to the network processors, and in order to calculate the actual percentage composition, the Traffic Shaper makes use of the state information stored in Scratchpad. It performs a number of Scratchpad memory accesses that is equal to one more than the number of the active aggregate profiles. One memory access is required per profile to read the length of each aggregate profile. Additional memory access is required to read the combined length of all aggregate profiles. Therefore, its performance is proportional to the number of the active aggregate profiles.

5.3.2 AWA Traffic Shaper Performance

As described before, the traffic shaper uses the unallocated bandwidth from the profiles that are not using their assigned bandwidth and reassigns it to the profiles that need it. Figure 5.5 shows an example of the traffic shaper handling the highest priority. It can be seen that the Traffic shaper always finds middle "ground" between the actual traffic percentage and the assigned bandwidth. Because Profile 0 does not have enough traffic to use, the traffic shaper strips off the bandwidth assigned to profile 0.

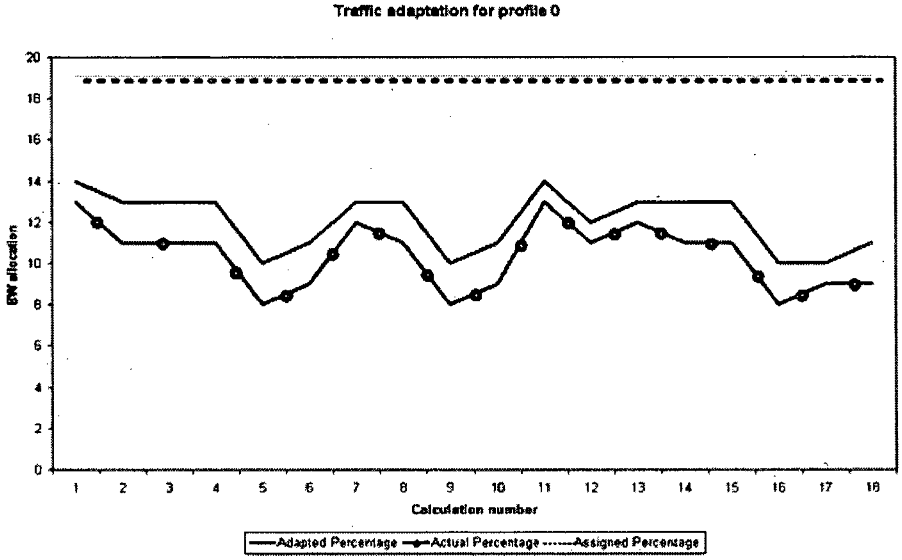


Figure 5.5: Adaptive Weight Adjustment for profile 0

Figure 5.6 shows an example of the adaptive weight adjustment handling the lowest priority profile. It can be seen that the Traffic shaper always finds middle "ground" between the actual traffic percentage and the assigned bandwidth. Because profile 8 has a traffic level that is much higher than it can accommodate within its share, the traffic assigns more bandwidth to the "needy" profile 8. The extra bandwidth was taken earlier from the unused bandwidth by other profile like profile 0.

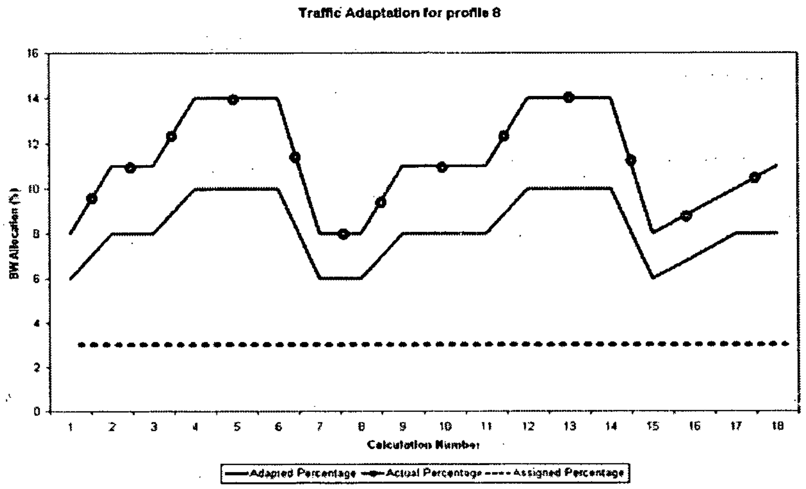


Figure 5.6: Adaptive Weight Adjustment for profile 8

Figure 5.7 shows the bandwidth allocation pattern for profile 4, which has a medium priority compared to profile 0 and profile 8. As shown in the figure, profile 4 has been provided less bandwidth than the assigned value because its actual traffic is low.

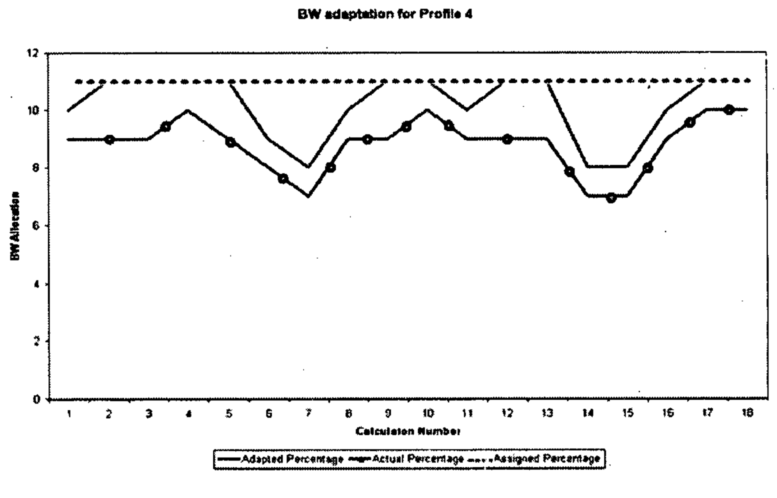


Figure 5.7: Adaptive Weight Adjustment for profile 4

However, the Traffic shaper ensures that this profile does not get more bandwidth than the assigned value when the actual traffic of the profile is less than the assigned value.

5.4 LCWFQ Scheduler

5.4.1 Assumptions and Notations

- The line rate (wire speed) is known to be a constant R bits per second.
- The total number of active aggregate profiles (N_p) is known.
- The elements (packets) of the aggregate queue are sorted according to their arrival time in the queue.
- The rate for every active aggregate profile (r_i) is specified as a percentage of the line rate, where r is the percentage bandwidth allocation of the i^{th} profile.

5.4.2 The LCWFQ Name Origin

The "Lowest Credit" name is driven from your bank credit card statement value when it is negative, it means the bank owes you money and when it is positive, it means you owe the bank. Assume that all cardholders are the only owners of the bank and assume simple bank operation that when you owe the bank, you owe each cardholder a percentage of your debts equal to the share of the cardholder.

5.4.3 LCWFQ Concept

Similarly, the LCWFQ scheduler serves all profiles as if all the profiles own the scheduler and each profile owns a share equal to the assigned bandwidth percentage defined by the SLA for that profile. When a packet is served from a profile for a certain time t, the served profile owes each of the active non-served profiles a share of the time t

to wait. Each of the non-served profiles gets a share equal to the time t multiplied by the assigned bandwidth percentage for that non-served profile.

The credit value C_i for the i^{th} profile that has just been scheduled and is calculated according to the formula $C_i = C_i -1 + L_i * (1- r_i/100)$. This is the bandwidth that the current profile "owes" to the other profiles. The credit value for the other profiles is calculated according to the formula $C_i = C_i -1 - L_i * r_i/100$. The right operand calculates the share of the total bandwidth that is owed to this profile by the profile of the transmitted packet. The whole expression evaluates the net credit value for the profile.

After the credit values for every profile is updated, the scheduler goes back to fetch the record for the next packet in the next profile. If there are no packets in the queue, the scheduler resets the previous credit values. If there is only one packet in the queue, the scheduler updates the previous credit to the new calculated value. If there is more than one packet in the queue and if all the packets have the same profile, the packets will be scheduled according to their order in queue. Within the same profile, the first packet encountered first is the first to be scheduled.

However, if the packets in the queue have different profiles, the scheduler will select the packet that has a profile of the lowest credit value. Again, the credit value of each profile is calculated accordingly. A sample scheduler calculation for the scheduler is shown table 5.3. For simplicity, it is assumed that all packets have the same length of 1500 bytes.

SCHEDULED PACKETS

Packet	Rate	10%	20%	30%	40%
Length	Profile	Credit 1	Credit 2	Credit 3	Credit 4
		0	0	0	0
1500	4	-150	-300	-450	900
1500	3	-300	-600	600	300
1500	2	-600	600	150	-300
1500	1	750	300	-300	-900
1500	4	600	0	-750	0
1500	3	450	-300	300	-600
1500	4	300	-600	-150	300
1500	2	150	600	-600	-300
1500	3	0	300	450	-900
1500	4	-150	0	0	0
1500	1	1200	-300	-450	-600
1500	4	1050	-600	-900	300
1500	3	900	-900	150	-300
1500	2	750	300	-300	-900
1500	4	600	0	-750	0

Table 5.3: Numerical example of the scheduler

5.4.4 LCWFQ Operation

The scheduler serves only one packet in a full round. The scheduler starts the round by checking the credit value of all active profiles and serves a packet from the profile that has the lowest credit. A credit is defined as the assigned bandwidth percentage that a served profile owes to other non-served profiles. A negative credit value for a profile indicates that the scheduler owes the profile a service. If the system owes multiple profiles a service, the system serves the profile with the lowest credit value first.

Initially a credit of zero is assigned to each profile. As packet(s) arrive in the queue, the scheduler selects a packet that has the lowest credit and places it in the corresponding egress queue then updates the credit value for all profiles. The credit value Ci for the ith profile that has just been scheduled is given by

$$C_i = C_{i-1} + L_i * (1 - \frac{r_i}{100})$$

Following the logic, for the non-served profiles, the credit value Cj for the jth is given by

$$C_j = C_{j-1} - L_i \frac{r_j}{100}, \quad i \neq j$$

The right operand evaluates the share of the total bandwidth that is owed to the profile. After the credit value for each profile is updated, the scheduler starts the next round and fetches next record for the next packet. If there are no packets in the queue, the scheduler resets the previous credit values.

If, during the scheduling process, the LCWFQ scheduler finds that any two or more aggregate profiles have identical and lowest credit values, it selects a packet that has an aggregate profile with the highest assigned percentage. However, if all of the lowest credit profiles have identical assigned percentages, it schedules the packets in a simple round robin fashion.

LCWFQ always operates with O(1) work complexity and, unlike DRR and PDRR, does not require the prior knowledge of the maximum size of a packet that eventually arrives in the system. All profiles are considered during every scheduling cycle and only one of them qualifies to have a packet scheduled. In addition, the scheduling process is independent of the status of the input queue. That is, it performs steadily whether the queue is backlogged or not. In the case when a profile is empty, it accumulates credit only to a pre-specified level. In addition, a profile can only owe a pre-specified level of credit to the remaining active profiles.

5.5 Performance Analysis of LCWFQ

5.5.1 Computational Complexity

The input queue to the scheduler is organized such that it has as many queues as the number of profiles. The scheduler, the traffic shaper, and the profiler share a data structure that describes the queue. The description of the queue contains the location of the head of each sub-queue. The profiler adds a packet at the end of the queue and the scheduler removes a packet from the beginning of the queue. Therefore, the scheduler does not spend any time searching for a packet within the queue. That is why the per packet work complexity of the LCWFQ scheduler is O(1). The scheduler performance does not depend upon knowing the maximum packet size that possibly arrives in the system. Furthermore, there is no hashing involved to queue packets according to their profiles. Each aggregate profile is assigned its own queue and newly arriving packets are appended at the end of the queue. This renders the queuing mechanism to be simple and low processing cost.

5.5.2 Startup Latency

The worst-case situation is when all aggregate profiles are active at the same time. Assume a case when the router restarts or when many packets arrive at the same time from multiple input ports and exit from one single output port. Also, assume that these packets belong to deferent profiles. In this circumstance, the LCWFQ scheduling algorithm selects the packet that has a profile of the highest bandwidth percentage assignment. The worst-case scenario is when a profile i has the lowest assigned bandwidth. The packet that belongs to the profile in question will wait until its credit value becomes the lowest. This happens after $(N_P - 1)$ aggregate profiles are served where N_P is the number of the active profiles. Assume that m is the size (in bits) of the largest packet served during the execution of one scheduling cycle. It implies that, at most, $(N_P - 1)*$ m bits are sent before the first bit of the profile is sent. Then the maximum latency experienced by an aggregate profile in a startup cycle is

$$D_{LCWFQ} \leq \frac{(N_P - 1)m}{R}$$

5.6 Service Level Agreement (SLA)

The ISP may provide three types of services called Premium, Normal, and Light. Subscribers under the same type of service share the same aggregate profile. Assume the Normal service is an average type of service and the Premium service has higher priority and higher bandwidth allocation than the Normal service. Also, assume the Light service is a low speed service with a lower priority than the Normal and Premium Services.

Suppose that the premium SLA (profile 1) value is 60 (like $60 a month), the normal SLA (profile 2) value is 30, and the light SLA (profile 3) value is 10. Assume also, there is only one common output port channel that is used to carry the traffic from all subscribers. Then the fair percentage of the output port usage should be 60%, 30%, and 10% respectively.

However, if there are 20 subscribers that have a premium SLA of profile1-1, 10 clients with SLA of profile-2 and 10 clients with SLA of profile-3 and all of them uses the common channel simultaneously. Then profile-1 subscribers would require 1200 points value of the port assigned bandwidth percentage, while profile-2 Subscribers would require 300 points value of the bandwidth, and finally profile-3 clients would require 100 points value of the bandwidth. In this case, the bandwidth allocations can be adjusted so that it reflects the weight of the original percentage assignments. The results are summarized below.

SLA Value	Symbolic Representation	Profile1 BW_1= 60%	Profile2 BW_2= 30%	Profile3 BW_3= 10%
# Of subscribers per SLA	(N_{C_i})	20 Subscribers	10 Subscribers	10 Subscribers
Relative BW Requirement	$(BW_i * N_{C_i})$	60 x 20 = 1200	10 x 30 = 300	10 x 10 = 100
BW Assignment **Per Aggregate profile**	$(\frac{BW_i * N_{C_i}}{\sum BW_i * N_{C_i}})*100$	75.00 %	18.75 %	6.25 %
BW Share per Client	$(\frac{BW_i}{\sum BW_i * N_c})*100$	3.75% per subscriber	1.875% per subscriber	0.625 % per subscriber

Table 5.4: Illustration of Profiling based on a Service Level Agreement

The SLA manager accepts the SLA configurations from the administrator. Individual SLA specifies the maximum delay and minimum transmit rate then translated into assigned bandwidth BW for each aggregate profile and maximum delay aggregate profiles whenever required.

The shaper readjusts the assigned bandwidth and reports over-usage using the maximum delay. For example, for a certain profile P_i, if the line speed is 10Mb/s and the SLA for P_i is 10% guaranteed bandwidth and the maximum delay guaranteed is 250ms, there will be number of issues to consider. In order to guarantee the maximum delay for a certain SLA, the SLA should include profile rate ratio, $r_{MaxDelay}$, of

$$r_{MaxDelay} = \frac{LP_i}{TP_i x R}$$

Where R is the line rate, where TP_i is the Max delay for a profile i, and where LP_i is the length of profile i in the input queue.

For example if the queue depth for P_i in the scheduler input queue is equal to a burst of 500K bits, which needs 50 ms to be transmitted, in this case, there is no issue in conflict with the SLA. However, assume that the total traffic in the input queue from P_i and other profiles is equal to 10M bits, which equal to one second of transmission. If 10% of BW is assigned for the P_i as per the SLA, then the maximum delay will reach about 500ms for the last packet sent from P_i. In order to comply with the SLA Maximum delay then the SLA manger must increase the bandwidth assigned to P_i from 10% to 20% BW. That means finishing the P_i traffic transmission within 250ms. After 250ms of transmission, there will be no more packets from P_i to transmit, so the traffic shaper redistributes the unused 20% bandwidth from P_i profile to the other profiles.

If the queue of P_i is a burst of 5M bits which needs 500ms to transmit, then there is an issue of conflict with the SLA as P_i should have been assigned at least 50% the bandwidth in order to grantee the maximum delay.

5.7 FIFO-Effect

The Scheduler is the module that determines the order with which incoming packets are disposed to the Egress module for transmission. The egress module always serves the packets on a FIFO basis. Since there is a speed mismatch between the Profiler, the Scheduler and the Egress module, regardless of the scheme the scheduler implements, the scheduler may always operate on a FIFO basis, due to a phenomenon termed here as the FIFO-Effect. The cause of this effect and its remedy are described hereinafter.

FIFO-Effect is defined as a phenomenon whereby a scheduler implementing a non-FIFO algorithm inadvertently behaves as a FIFO scheduler. This occurs because the scheduler operates in between the forwarder module and the Egress module and it is much faster than both the egress process and the forwarder process. Since the scheduler works with packet records, and not with actual packets, it takes the same amount of time to schedule any packet regardless of the length. So, the speed of the scheduler is determined only by the computational limitations of the hardware platform upon which it is running.

Definition: The Profiler/Forwarder processing time, T_P, is the time required processing a packet to determine the route of the packet and its profile, to compile the relevant packet descriptor record, and place the record in the scheduler queue.

Definition: The Scheduler processing time, T_S, is the time the scheduler required reading one packet record, process the record according to the algorithm it implements, and place the record in the output queue.

The Scheduler Processing Time is constant for every packet. Every record within the scheduler queue contains the length of the packet, the packet pointer (the address of the packet within memory) and the profile of the packet. Furthermore, any scheduling algorithm performs only a finite number of actions such as multiplications, additions, and comparisons. These operations are performed on every record within the scheduler queue. Therefore, the scheduler processing time is approximately constant for every packet.

Let TP be the packet processing time of the profiler/forwarder unit and TS be the time the scheduler requires to schedule one packet. The maximum number of packets arriving in the scheduler queue, NS, during TS is given by,

$$N_S = \left\lfloor \frac{T_S}{T_P} \right\rfloor \qquad , N_S \in I.$$

If the scheduler is faster than or as fast as the profiler/forwarder, it operates as a FIFO no matter which scheduling algorithm it implements. Let NS(t) be the number of packets within the scheduler queue at arbitrary time and let nS be the number of packets that have been scheduled during a time period of t. Therefore,

$$t = n_S T_S$$

During the same period, the number of packets that have been profiled and placed in the scheduler, np is given by,

$$n_P = n_S \left\lfloor \frac{T_S}{T_P} \right\rfloor$$

The total number of packets within the queue is therefore given by,

$$N_S(t) = N_S(n_S T_S) = n_P - n_S = n_S \left\lfloor \frac{T_S}{T_P} \right\rfloor - n_S = n_S (\left\lfloor \frac{T_S}{T_P} \right\rfloor - 1).$$

N_S(t) will have a positive value only when TS > TP. Otherwise, it is zero or negative. However, the number of packets cannot be negative. It only means that there are no packets with in the queue. Therefore, it can be concluded that if the scheduler operates faster than or as fast as the profiler/forwarder, there will only be a maximum number of one packet within the queue. Thus, the scheduler decides to schedule this packet because it is the only packet with in the queue. Hence, packets are dispatched to the Egress module in the same order as they have arrived.

The implication of this behavior is that if the line rate is such that the transmission time of a single packet is greater than the *Scheduler Processing Time*, any backlog that may exist at the input of the Profiler/Forwarder is invisible to the Scheduler. To avoid this problem, the scheduler must wait for duration of time, T_W, given by,

$$T_W = \frac{L_E}{R}$$

Where L_E is the combined length of all the packets in the output queue of the scheduler and R is the line rate, then the existing backlog will be transparent to the scheduler and hence the scheduler will be able to dispatch packets according to the algorithm that it implements. Specifically, implementation of the scheduler on high performance platforms such as the IXP2400 NP incorporates such phenomena. The FIFO-Effect is illustrated hereinafter.

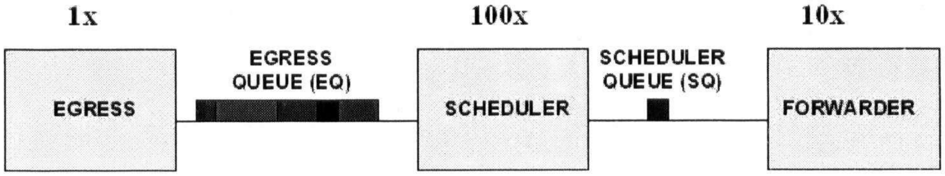


Figure 5.8 Diagram showing the FIFO Effect

Suppose that the scheduler process works 100 times faster than the Egress process and 10 times faster than the forwarding process. This implies that during the time the Egress module transmits one packet, the scheduler can schedule 100 packets and places them in the Egress Queue. In the same period, the forwarder can place 10 packets in the scheduler queue. Since the Egress module transmits packets in the order of their arrival in its queue and the scheduler is disposing packets one at a time, the whole process represents a FIFO scheme no matter which scheduling algorithm is implemented by the scheduler.

5.8 FIFO-Effect Suppression

To counter this effect and to ensure that packets are scheduled as per the algorithm implemented by the scheduler, we propose that the scheduler has to take into consideration the Egress process time required to transmit all the packets found in the Egress Queue.

5.8.1 FIFO-Effect Suppression Mechanism

Definition: The scheduler is said to be *Serving* when it is busy transferring packets from scheduler input queue and disposing packets into the Egress Queue.

Definition: *Service Window* is defined as the time interval during which the scheduler is serving.

Let $\boldsymbol{l}$ be the total length of packets that have been scheduled during the *Service Window*. For a line rate of $\boldsymbol{R}$ bits per second, it takes the Egress module a transmission time given by the following formula:

$$T_{EQ} = \frac{l}{R} \text{ in seconds}$$

Where T_{EQ} is defined as the time required for the Egress module to transmit all the packets in the Egress Queue

During this time, the scheduler should not send packets to the Egress Queue because the Egress Module is already busy and any more packets sent to the Egress module will be queued. During this time, packets should be accumulated in the scheduler input queue. In this way, the scheduler behaves more towards the expected scheduling scheme, which in this case is LCWFQ.

Therefore, the FIFO-Effect suppression process forces the scheduler to wait for a length of time equal to T_{EQ} before scheduling the next round of packets. The scheduler, thus, will be able to see more packets and hence schedule the packets according to the bandwidth quota of the profiles. If there is a continuous flow of packets, the number of packets that the scheduler sees within the queue increases. As time progresses, the scheduler will be able to see more and more packets within the queue and hence recalculates the bandwidth accordingly. The more packets the scheduler sees, the more accurate the fairness will be. The detail of the operation of the scheduler is illustrated below.

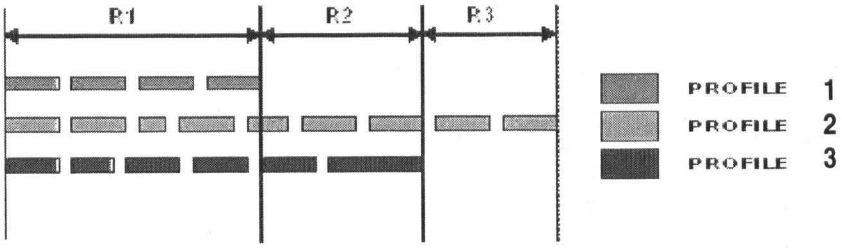


Figure 5.9 Illustration of the Scheduling and the FIFO-Effect

Simple example to explain service windows R1, R2 and R3:

Suppose there are three profiles; profile 1 contains only 4 packets, profile 2 contains 9 packets, and finally profile 3 contains 6 packets. For simplicity, assume that all profiles assigned the same bandwidth percentage such that the scheduler will behave as a Round Robin scheduler. Assume that no more packets arrived for a period long enough to dispose all packets in the queue.

In the first stage of packet scheduling, the scheduler continues to schedule packets until the fifth round. At this instant, the scheduler is ready to schedule packets from profile-1, but there is no packet to serve. At this point, the scheduler stops to fork the shaper to recalculate the new bandwidth percentage distribution. The shaper calculates the time, T_{EQ}, required by the egress to finish transmitting the scheduled packets and waits for that time. During T_{EQ} more packets for profile-1 may arrive. Even if no packets arrive, as assumed in this example, a new bandwidth distribution is calculated at the end of T_{EQ}. R1 here is defined as a service window where the scheduler is using fixed pre-calculated bandwidth distribution. Then another window R2 starts and the previous scenario repeated again until the second profile finishes, which is after R2 period. At the end of R2, the scheduler realizes that there are no more packets for profile-3 to serve. At this time, the scheduler again enters into a wait state, recalculates the bandwidth distribution to include any new arrived packets, and eliminates any finished profiles. Finally, Window R3 starts and repeats the same process until the current Window R3 expires or profile-2 runs out of packets, whichever comes first.

Once the egress process finishes transmission, the traffic shaper starts to calculate the adapted bandwidth quota for each profile and then the scheduler dispatches the packets from the profiled queues. The scheduler stops scheduling if it runs into a Silent Gap or if any of the active profiles is due to be scheduled but do not have a packet as determined by the scheduling algorithm.

The following example illustrates the procedure described above. Suppose that the Egress Queue is initially empty and let the line rate be 10Mb/s. Thus, to transmit the longest packet, say 1540 bytes, the Egress module requires a total time T_{EQ} shown below.

$$T_{EQ} = \frac{1540*8 \ bits}{10{,}000{,}000 \frac{bits}{s}} = 1.232 \ ms$$

During 1.232 ms, the scheduler enters into a wait state during which the Profiler can place 10 packets in the profiled queues into the input queues of the scheduler. T_{EQ} is, therefore, the first waiting time and the next Packet Disposition Time Window. When *1.232 ms* is elapsed after the scheduler enters in to the wait state, the scheduler requests new bandwidth allocation percentages from the Traffic Shaper. The Shaper, then, calculates the new adapted percentages based on the current profiled queues, marks the end of each input queue, and passes the BW information to the scheduler.

Consequently, the scheduler starts disposing packets based on the received BW allocation information. It continues to do so until one of the profiled queues runs out of packets. During the process newly arriving packets records are placed in the scheduler input queue but will not be recognized by the scheduler. The scheduler assumes an input queue is empty when it finishes scheduling all the packets before running into the mark. After scheduling the 10 packets presented to it by the profiler, the scheduler enters into a wait state whose duration is determined by

$$T_{EQ} = \frac{Total\ Length\ of\ Scheduled\ Packets}{Line\ Rate} = \frac{l}{R}.$$

Suppose that the scheduled packets are voice packets. That is, the length of each packet is 64 bytes. Therefore, the new waiting time is calculated by

$$T_{EQ} = \frac{10*64*8*10\ bits}{10{,}000{,}000\frac{bits}{s}} = 5.12\ ms$$

If, in the worst case, all the packets were the longest (1540 bytes), the waiting time would be calculated by

$$T_{EQ} = \frac{10*1540*8*10\ bits}{10{,}000{,}000\frac{bits}{s}} = 123.2\ ms$$

Therefore, the scheduler waits for the calculated period before it starts the scheduling process again. During this period, more packets are accumulated. The adaptive weight adjustment module in the scheduler determines how long to wait before scheduling packets, then enforce BW allocation assignments while keeping the egress module busy transmitting packets into the external network.

Another way of FIFO-Effect Suppression is to force the Traffic Shaper to perform its bandwidth allocation calculations after scheduling every packet or, in general, after scheduling a number of packets whose transmission process takes an amount of time that is sufficient for the traffic shaper to calculate a new bandwidth allocation quota for each profile. Thus, following the procedure described above, the scheduler waits T_{EQ} long, calculates the percentage based on the existing input profiled queues and the scheduled the packet from the appropriate profile. This optimum solution, therefore, limits the Egress Queue Depth to one Packet. The proposed traffic shaper takes care of this relationship in performing two functions. First, it serves the burst even if it is conflicting with the SLA whenever possible and meanwhile reports the over-usage to the SLA manager [16]. Second, it fairly serves P_i traffic in case of backlogged queue coexists from the other profiles.

6 Simulation and Results

6.1 Simulation Overview

In order to simulate the scheduler and the traffic shaper, a C++ object oriented program was designed and implemented to generate the packet traffic and simulate the operations of the Scheduler and the Traffic shaper [20]. This is accomplished in two steps: Traffic generation and operational simulation. Finally, a simulation of FIFO effect suppression is presented.

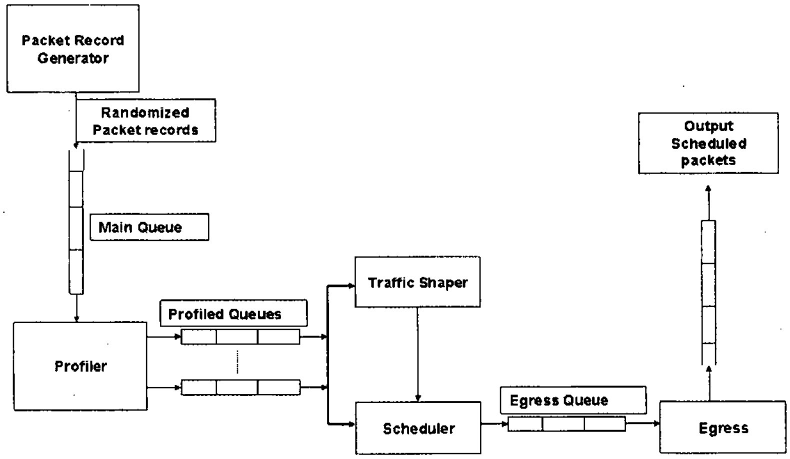


Figure 6.1 The Simulator Schematic Diagram

Main Queue: Simulates an input queue where incoming packets are stored by order of arrival.

Profiler: Assigns profiles to incoming packets and places them in the respective queue

Profiled Queue: Simulates queues of pointers of packets that are classified per their profile.

Traffic Shaper: Automatically adjusts the BW allocation of each profile if the option is selected

Scheduler: Schedules packets from the profiled queues according to the algorithm it implements

Egress Queue: Simulates queues of pointers of the scheduled packets and ready for transmission.

Egress: It represents the module that transmits the scheduled packets.

6.2 Traffic Generation

Traffic generation is defined as the random generation of packet records called profile, and packet length. This is because the profile and the packet length are the two attributes that the traffic shaper and the scheduler need in order to modify the percentage bandwidth quota or dispatch the packet respectively. Randomization was important in order to mimic a real traffic situation, which is realized using the Bernoulli's process described below. Let us first define some terms used in the rest of this chapter.

Definition: *Burst Span, BS,* is defined as the period of time during which packets are assumed to arrive with a very little time difference.

Definition: *Silent Gap, SG,* is defined as duration of time in which no packets arrive.

Definition: *Simulation Window, SW,* is defined as a period comprised of two or more Burst Spans and one or more Silent Gaps.

Definition: *Queue depth, Q_D,* is defined as the total length of all Burst Spans and Silent Gaps expressed as a percentage of the line rate.

The input parameters for the traffic generator are Minimum and Maximum Burst Spans, Minimum and Maximum Silent Gaps, Simulation Window, and Queue Depth.

6.2.1 Traffic Generation Using the Bernoulli's Process

The packets are assumed to arrive in bursts separated by *Silent Gaps*. These *Silent Gaps* are randomized according to the following formula:

$$SG_i = R * MinimumSG + (rand()\ \%\ (MaximumSG - MinimumSG)),$$

Where SG_i is the length of the i^{th} Silent Gap, *R* is the line rate in bits per second,

MaximumSG is the maximum length of Silent Gap expressed in seconds and,

MinimumSG is the minimum Silent Gap expressed in seconds.

Similarly, the length of each burst is calculated by the following formula.

$$N_i = MinimumB + (rand() \ \% \ (MaximumB - MinimumB))$$

Where N_i is the length of the i^{th} burst,

MaximumB is the maximum burst length expressed in terms of number of packets,

MinimumB is the minimum burst length expressed in terms of number of packets.

The maximum length of the IP packet is 1540 bytes and a minimum of 20 bytes. In the simulation, these values define the Range, *Rg*. The Range is divided into N_I intervals. Each interval represents a packet of length, L_P, calculated by

$$L_P = \frac{Rg}{N_I}.$$

A Binomial probability distribution function quantifies the probability of a number of successes in a given number of trials that is described by the following formula.

$$P(X = x) = {}_nC_x * p^x * (1-p)^{n-x}$$

Where X is a random variable representing the number of successes in *n* trials and *x* is the number of successes. In the simulation, this probability determines the percentage of packets whose length, l_x, is $x * \frac{Rg}{N_I}$ and the probability distribution will take the form shown in the following formula.

$$P(X = x) = {}_{N_I}C_x * p^x * (1-p)^{N_I - x}$$

The simulator converts every input parameter into an equivalent number of packets with a variable length. Therefore, given the total number of packets N_p that need to be generated, the number of packets in each interval, n_x, is calculated using the following formula.

$$n_x = N_p * P(X = x) = N_p * {}_{N_I}C_x * p^x * (1-p)^{N_I - x}$$

In addition, these packets have a length of l_x.

In the simulator, packets are generated for every interval, and hence, N_P represents the number of packets per interval, N_i, calculated above. However, the profiles are generated by the following C++ code shown below.

```
srand(time(NULL));
Profile = rand % NumberOfProfiles;
```

Some screenshots are presented in Appendix B where it shows the details of setup parameters described before.

6.2.2 Results of Traffic Generation

Figure 6.2 shows the graph of packet length generation for packets arriving at the scheduler in four bursts and three silent gaps. It can be seen from the graph that the packet length distribution per interval is random. While the variation of the length within every burst follows Bernoulli's distribution, it is randomized to vary the arrival of packets of same length so that generation scheme is appropriate for the simulation. The long spikes indicate the lengths of the silent gaps expressed in equivalent number of bits.

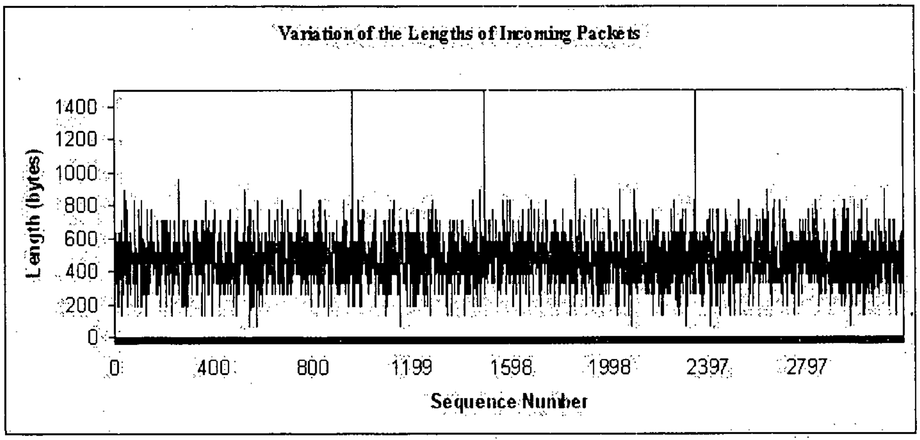


Figure 6.2 Sample of Generated Packets Length Variation

Figure 6.3 shows the distribution of packets within each profile. Note that the number of packets shown in the figure is 3,500,000 packets. Again, it shows four bursts and three silent gaps.

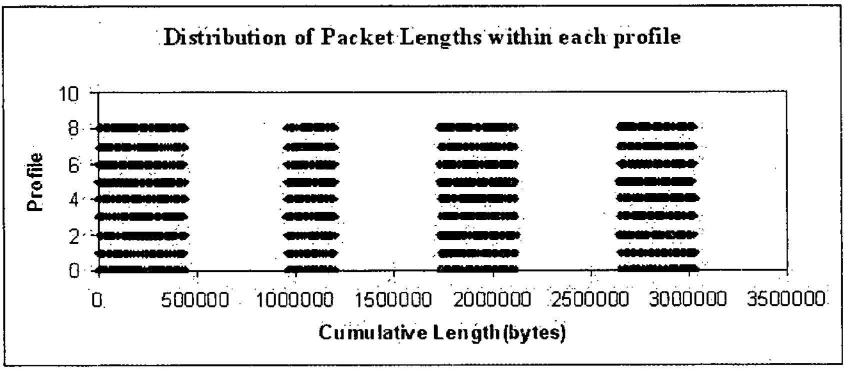


Figure 6.3 Generated packets arrivals per profile for a case of four bursts

6.3 Simulation

The simulation is done in two stages: Scheduling with fixed bandwidth and Scheduling with adapted bandwidth.

6.3.1 Scheduling with Fixed Bandwidth

The performance of the FIFO, SRR, DRR and LCWFQ scheduling algorithms for identical traffic input of two bursts separated by one silent gap and a fixed bandwidth assignment for every profile, shown in Table 6.1, is shown in Figure 6.4.

Profile	0	1	2	3	4	5	6	7	8
BW Assignment (%)	19	17	13	11	9	7	3	2	1

Table 6.1: Bandwidth quota assignment assumed by the SLA Manager

Figure 6.4 shows the average delay for the four scheduling algorithms. It can be seen from the figure that LCWFQ prioritize, and discriminates the profiles according to their bandwidth assignments. Since profile 0 has the highest bandwidth quota, it incurs the lowest delay. On the other hand, profile 8 incurs the highest delay because it has the lowest bandwidth quota, and hence, the lowest priority.

Figure 6.5 shows the maximum delay for the same and further clarifies this assertion. The figure reaffirms the ability of the LCWFQ algorithm to discriminate between different profiles according their priority. It should be noted that the performance of the LCWFQ is a little bit better in the average delay, while it provides the same performance on the maximum delay. The LCWFQ performs the same during the bursts and the silent gaps. While the empty queues and silent gaps cause errors to the DRR.

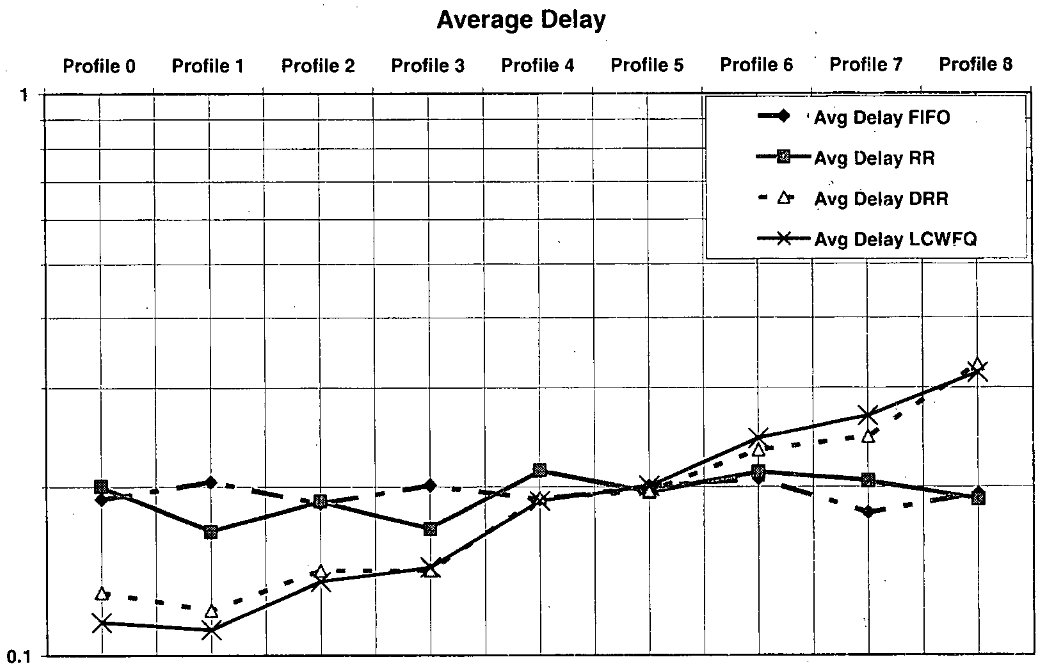


Figure 6.4 The average delay with fixed B/W assigned

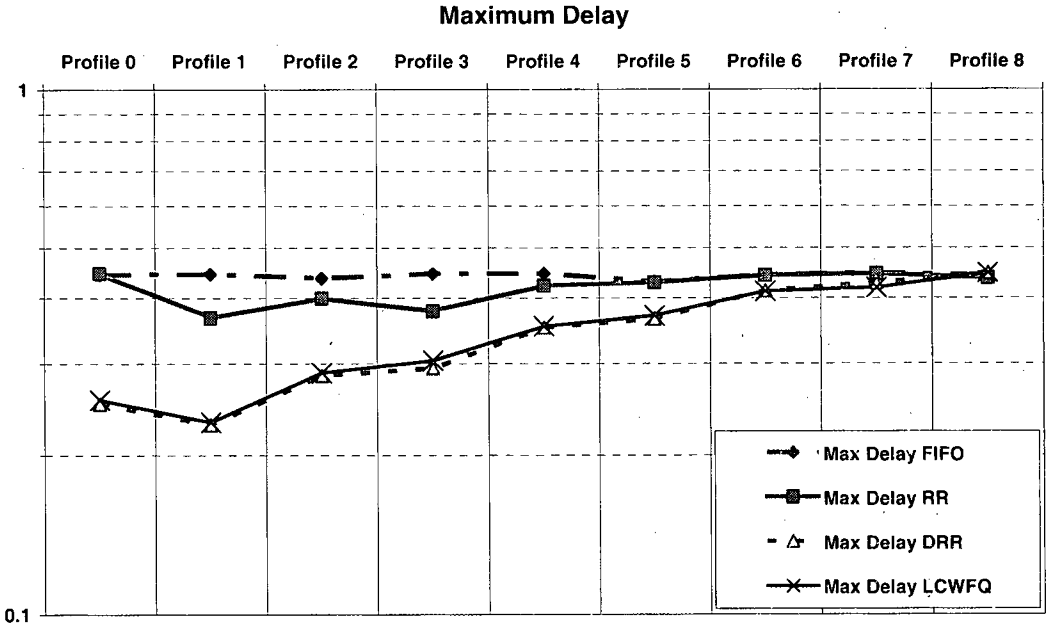


Figure 6.5 The maximum delay with fixed B/W assigned

6.3.2 Scheduling with Adapted Weight Adjustment

Scheduling with adapted weight adjustment is performed with the help of the Traffic Shaper. As described in chapter 3, the Traffic Shaper adjusts the bandwidth allocation toward the actual incoming traffic composition to distribute allocated but unused bandwidth of profiles that have higher bandwidth quota and have lower traffic to those profiles that have lower bandwidth quota and have higher traffic. The original bandwidth assignment is used as a reference to determine the relative priority of the profiles. Figures 6.6, and 6.7 show the average delay per profile, and maximum delay comparisons.

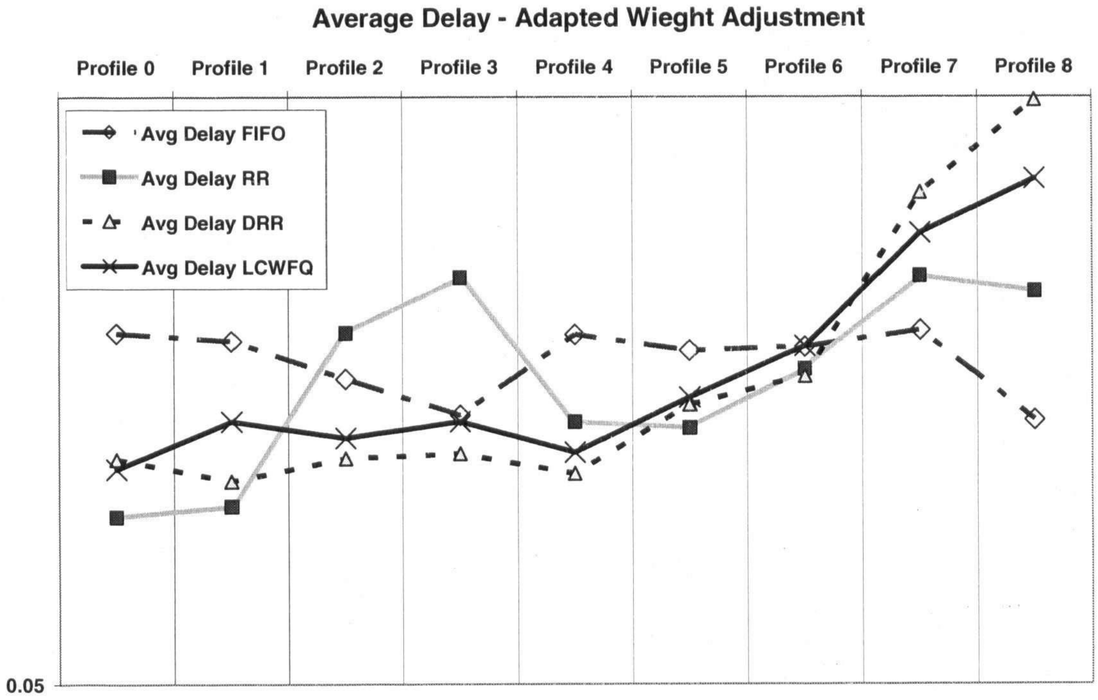


Figure 6.6: Average delay with adapted weight assignment

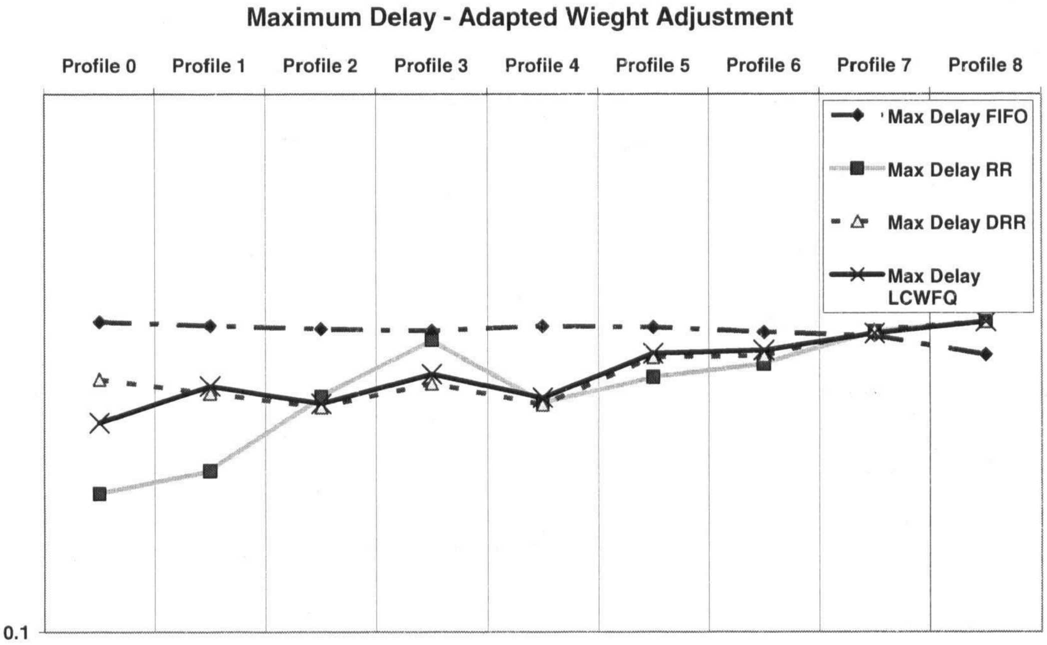


Figure 6.7 The maximum delay with adapted weight assignment

6.3.3 Fairness Index diagrams

The fairness of the four algorithms under consideration is calculated and the percentage allocation of the bandwidth is plotted against the sequence number of the packets as shown in Figures 6.8, 6.9, 6.10 and 6.11. The straight lines indicate the assigned bandwidth while the others show the actual bandwidth allocation as determined by the corresponding scheduling algorithm [21][22].

The graphs show that the FIFO and the RR algorithms do not ensure fair sharing of bandwidth. The fairness of the FIFO is such that the output pattern is the same as the input pattern. Because of this, bursty profiles get unfair usage of the output link. That is, if the lowest profile has many packets that have arrived just before highest profile packets, then the highest profile will be treated unfairly because it will be forced to wait despite its status.

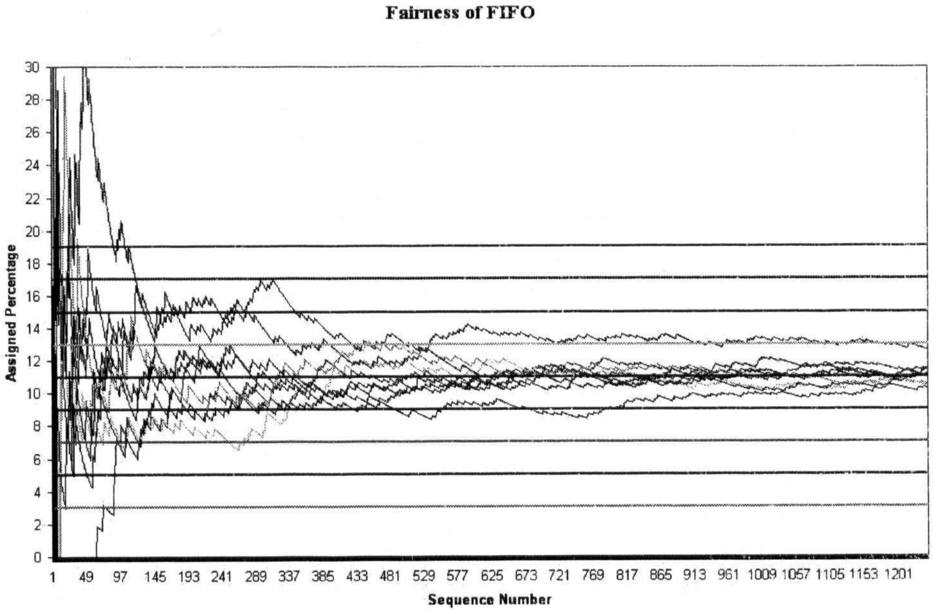


Figure 6.8 Fairness measurement of the FIFO scheduling algorithm

The Round Robin algorithm corrects the situation a little bit by scheduling packets from each profile that has packet in every round. However, RR suffers from the lack of an ability to enforce bandwidth allocation based on bandwidth quota. It is suitable only when the bandwidth is shared equally between all the profiles. Because RR inherently favors longer packets, bursty packets that are longer and that have lower profile get unfair advantage over higher profile packets that are shorter.

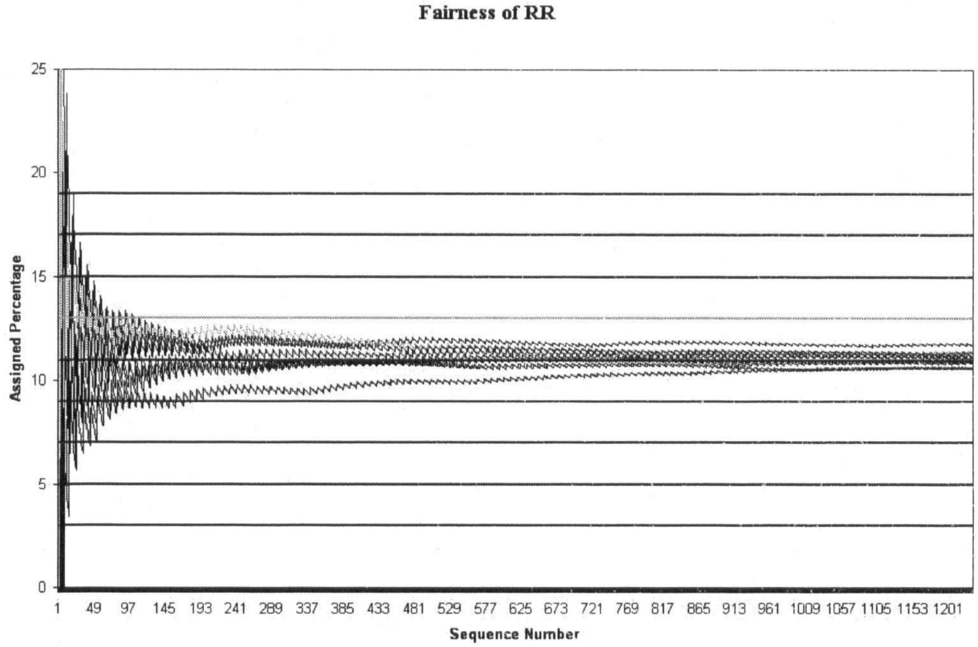


Figure 6.9 Fairness measurement of the RR scheduling algorithm

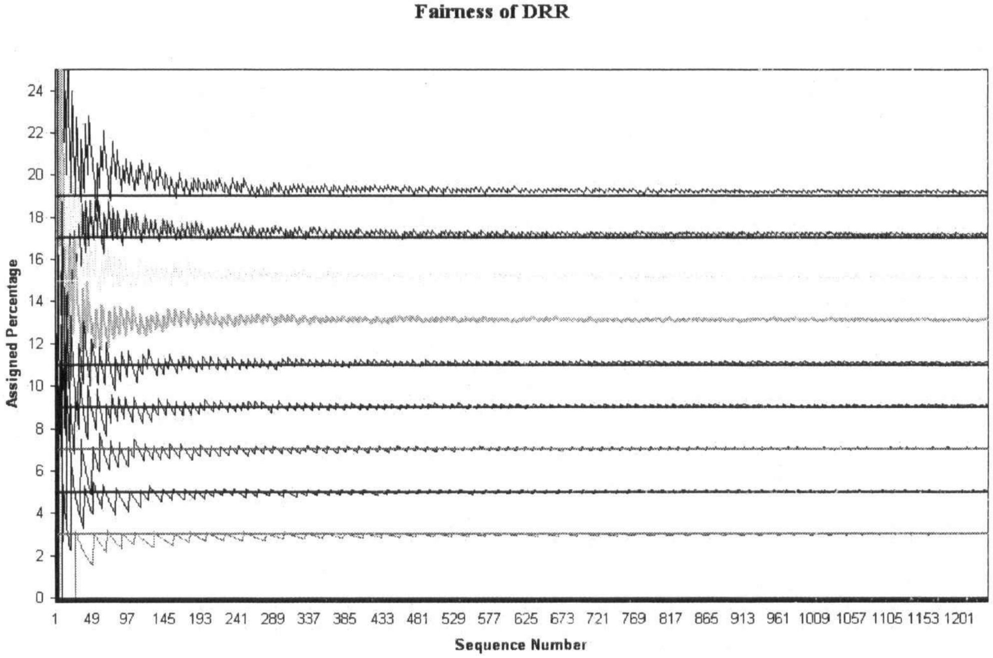


Figure 6.10 Fairness measurement of the DRR scheduling algorithm

Figures 6.8 to 6.11 also indicate that on the long run, the allocation of bandwidth between the profiles approach the assigned bandwidth only for DRR and LCWFQ algorithms. However, it is worth noting that LCWFQ offers relatively more stability than DRR and it does not suffer from the drawbacks of the DRR in that it behaves graciously when a specific profile has an empty queue. Figures 6.13 to 6.16 show the responses of the four scheduling algorithms for an input pattern shown in Figure 6.12.

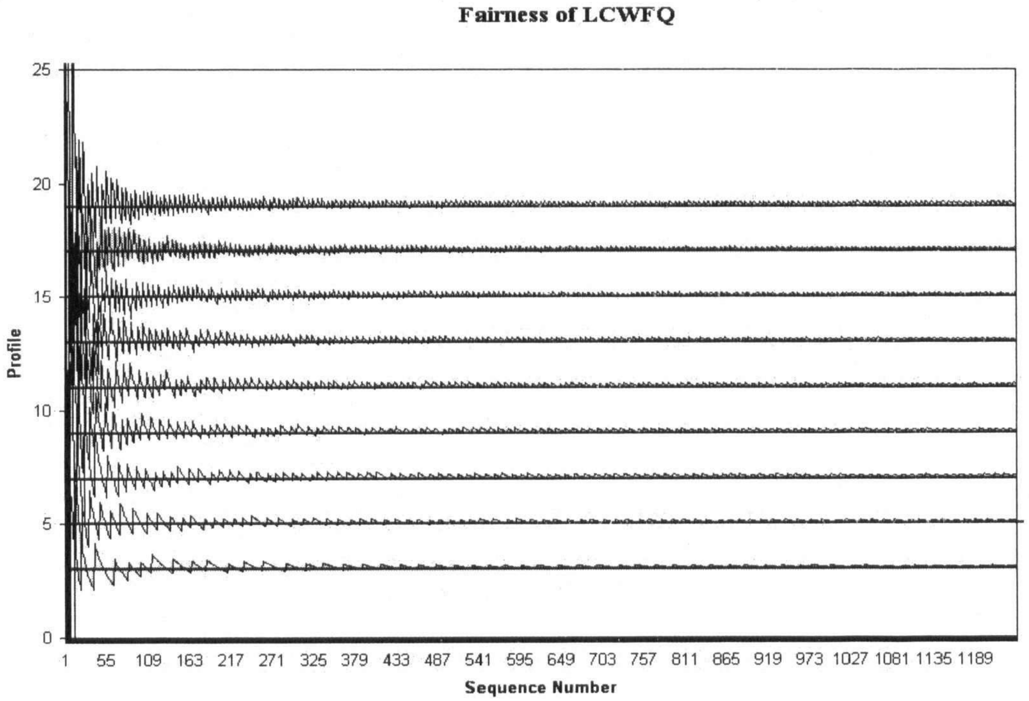


Figure 6.11 Fairness measurement of the LCWFQ scheduling algorithm

As the figures depict, the FIFO algorithm provides a chaotic response for a chaotic input. It does not provide any order to smooth the traffic or share the BW accordingly. The Round Robin algorithm provides a certain degree of order. However, it falls short of having the ability to perform fair bandwidth allocation. The Deficit Round Robin algorithm overcomes the disadvantages of the Round Robin scheduling algorithm by enhancing the order maintained by Round Robin to enforce a fair bandwidth allocation. The LCWFQ refines the DRR scheduler by providing more stability and better discriminative capability.

6.3.4 Simulation of the FIFO effect

As described in the previous chapter, FIFO effect is a situation in which a scheduler operates on a FIFO basis regardless of the algorithm it implements due to a speed mismatch between the module that supplies the packet records for the scheduler to work on and the scheduler itself. In order to simulate the effect, the scheduler uses a mechanism to control the number of packet records in the Egress queue.

Normally, the system operates with the principle that the scheduler is not required to schedule any packet until Egress module finishes transmitting the packets that are already scheduled. This will help the scheduler see as many packets as possible, and hence, enable it to distribute the bandwidth according to the assigned or adapted quota. In order to accomplish this, the scheduler keeps track of the combined length of all the scheduled packets that are disposed during the last scheduling cycle.

Therefore, if the number of packets the Egress queue can hold is not limited, or in other words, if the scheduler is allowed to schedule packets without waiting for the Egress module finishes transmitting the scheduled packets and if the scheduler is faster than the packet supplier module, then any scheduler operates in a FIFO mode. Figures 6.17 below demonstrate this effect.

The figure shows that the delay per packet is the same for all the profiles. The difference between the maximum delays of any two profiles is less than 0.006ms. If the number of packet records in the Egress queue increases, the aforementioned difference approaches to zero. This fact has been mathematically proven in chapter 3.

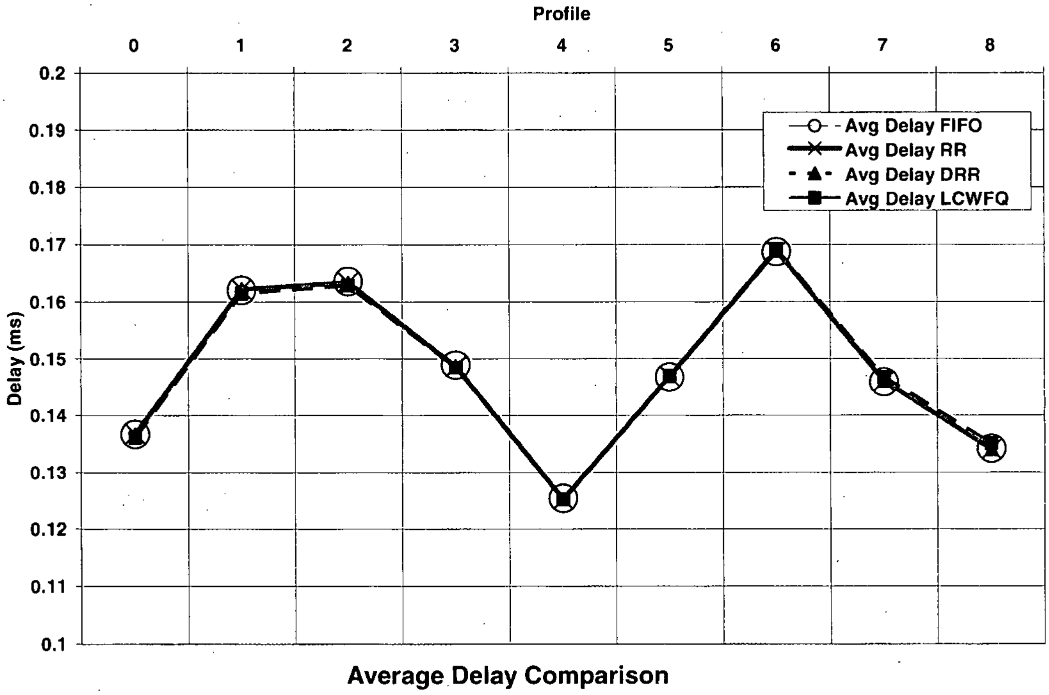


Figure 6.12 The Average delay when the Egress queue size is 1000 packets

6.4 Memory Access Overhead Estimates

It is assumed that the state information of all the queues implemented, as a shared memory between any two micro engines is small enough to fit in the Scratchpad memory. It is also assumed that the queue is implemented in the SRAM and the actual packet is stored in the DRAM. The Ingress module makes one SRAM write operation per packet to write the pointer of the memory in the SDRAM where the packet that has just been received is stored. It also reads the Scratchpad memory to read the pointer to head of the queue, called the non-proxy queue, and writes the same memory unit to update the new head of the queue.

The Profiler makes one Scratchpad memory read to get the tail pointer of the non-proxy queue. Using the tail pointer, it reads the non-proxy queue to retrieve the pointer to the packet. Afterwards, it writes the Scratchpad to update the tail pointer of the queue. It makes two-DRAM accesses to obtain the source and destination IP addresses. Using the source IP, it indexes into the SMM Active Catalogue to obtain the profile identification corresponding to the source.

It takes two extra Scratchpad memory accesses to read and update the head pointer of the Scheduler queue, which is shared between the Profiler and the Scheduler, four SRAM memory accesses to write the profile ID, the length of the packet the record represents the pointer to the actual packet and the pointer to the next record. It also makes two additional write accesses to the same memory unit to update the combined length of all the packets to which the profile of the packet under consideration belongs and to update the total length of all the packets represented by the records in the Scheduler queue. On top of this, the profiler needs to make at least two DRAM write operations to update the source IP, destination IP, source MAC, and destination MAC addresses. Assuming nine profiles, the Traffic shaper makes nine Scratchpads read memory accesses.

The Scheduler performs two Scratchpad memory accesses to read the tail of the pointer to use in removing a packet record from the queue corresponding to the profile that is being scheduled and to write the updated tail pointer. It makes four SRAM memory accesses to read from the Scheduler queue and another three memory accesses to the same memory unit to write into the Egress queue, which is the queue that serves as a shared memory region for the communication between the Scheduler and the Egress modules. The reason why the scheduler only requires two SRAM accesses is that the packet record written in the Egress queue is comprised of only the packet pointer and a pointer to the next record within the same queue.

Like the Scheduler, the Egress module performs two Scratchpad memory accesses to read the tail of Egress queue and to write the updated value back. In addition, the Egress module makes two SRAM memory accesses to read a packet record. The following table summarizes the above result. The IXP2400 Network processor speed is 600MHz.

	Scratchpad		**SRAM**		**DRAM**	
	Read (Cycle)	**Write (Cycle)**	**Read (Cycle)**	**Write (Cycle)**	**Read (Cycle)**	**Write (Cycle)**
Ingress	0	0	0	1	0	0
Profiler	4	4	2	4	2	2
Traffic Shaper	9	0	0	0	0	0
Scheduler	1	1	4	2	0	0
Egress	1	1	1	0	0	0
Total Num of Access	15	6	7	7	2	2
Approximate Latency	60		90		120	
Daley (µs)	0.1		0.15		0.2	
Total Daley per memory unit (µs)	2.1		2.1		4	
Total Delay (µs)	8.2					

Table 6.2: Typical Read/Write time for the router modules

7 Conclusion and Future Work

The thesis introduces number of additional and simple functions and methods to the router operation that process the received packets on early stages. In doing so, the rest of the router functions and modules turn into simple operations and require less processing resources compared to the conventional routers. The methods also fall under a new view of implementation to move away from the conventional forwarding and scheduling operations to seek the scalability and add more intelligence to handle complex operations in a simple way. By adding processing functions on early stages of the router, it saves processing time later. These proposed functions are the profiler, the Proxy Agent, the SLA manager, and weight adjustment traffic shaper. More depth in searching within the router for a record is also addressed by introducing the 3-level index method. The proposed functions are simple and low cost processes. Moreover, these functions make the other modules like the scheduler and the forwarder simpler and faster. From the simulation results, it is found that the 3-level index finds a route of any destination IP address and insures that the time required finding a record is always three memory accesses even if there is a million records. The proposed methods turn the router to be configurable to handle different types of multimedia packets through the inclusion of a proxy agent. The design includes a new traffic shaper method that adds another horizon to interpret the SLA and provide direct interface between a signed agreement and the active streams served by the router. The proposed LCWFQ scheduler has work load of O(1) and achieves a fairness index of 1 (One). The LCWFQ found to perform similar to the famous Deficit Round Robin (DRR) scheduling algorithm, but does not have the DRR weakness in dealing with the empty queues as the empty queue introduces errors in the DRR case. Unlike DRR, LCWFQ does not require the prior knowledge of the maximum size of a packet arrives in the system. The Service Level Agreement manager calculates bandwidth shares for all subscribers. Individual SLA specifies the maximum delay and bandwidth guarantee parameters, while the traffic shaper readjusts the assigned bandwidth and reports the over-usage cases.

There are three areas of research for future work. The first area is to design optimized memory index that can be used by all the modules in the router and the table structure. The second area is to introduce more algorithms in the area of the Adaptive weight adjustment. The third area is to design advanced SLA manager user interfaces.

8 Bibliography

[1] Intel® IXP2400 Network Processor Datasheet.

http://www.intel.com/design/network/datashts/30116411.pdf

[2] SIP Introduction by Jan Janak "A brief overview of SIP describing all important aspects of the "Session Initiation Protocol."

[3] IETF, RFC 3261 "SIP: Session Initiation Protocol"

[4] Understanding SIP by Dorgham Sisalem and Jiri Kuthan GMD FOKUS

[5] Narvaez, P.; Kai-Yeung Siu; Hong-Yi Tzeng, "Traffic Engineering with traditional Routing Protocols," IEEE/ACM Transactions on Networking, Volume: 9 Issue: 6, December 2001 pp. 706 -718.

[6] Ming-Yen Lin; Suh-Yin Lee, "Fast Discovery of Sequential Patterns through Memory Indexing and Database Partitioning" Journal of Information Science and Eengineering 21, pp. 109-128 (2005)

[7] Yuk Ho, "Application of Minimal Perfect Hashing in Main Memory Indexing", Massachusetts Institute of Technology, 1992

[8] Jussi Myllymaki and James Kaufman, "LOCUS: A Testbed for Dynamic Spatial Indexing" IBM Almaden Research Center, 2002

[9] D. C. Stephens, J. C. R. Bennett, and H. Zhang, "Implementing scheduling algorithms in high-speed networks," IEEE J. SelectedAreas in Commun., vol. 17, no. 6, pp. 1145–1158, Jun. 1999.

[10] D. Bertsekas and R. Gallager, Data Networks, Prentice Hall, New Jersey, 2nd edition, 1992.

[11] S. S. Kanhere, H. Sethu, and A. B. Parekh, "Fair and efficient packet scheduling using elastic round robin," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 324–336, Mar. 2002.

[12] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," IEEE/ACM Tran. Networking, vol. 4, no. 3, pp. 375–385, June 1996.

[13] J. C. R. Bennett and H. Zhang, "WF2Q: worst-case fair weighted fair queueing," in IEEE INFOCOM, Mar. 1996, pp. 120–128.

[14] P. Goyal, H. M. Vin, and H. Cheng, "Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks," IEEE Trans. Networking, vol. 5, no. 5, pp. 690–704, Oct. 1997.

[15] P.Goyal, H. M. Vin, and H. Haichen, "Start-Time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Netwirks," IEEE/ACM Trans. Networking, 5(5): 690-704, Oct. 1997.

[16] Naomi Karten, "Establishing Service Level Agreements", 2001, www.nkarten.com

[17] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," IEEE/ACM Trans. Networking, vol. 6, no. 2, pp. 175–185, Apr. 1998.

[18] J. A. Cobb, M. G. Gouda, and A. El-Nahas, "Time-shift scheduling – fair scheduling of flows in high-speed networks," IEEE Trans. Networking, vol. 6, no. 3, pp. 274–285, June 1998.

[19] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," IEEE/ACM Trans. Networking, vol. 6, no. 5, pp. 611–624, Oct. 1998.

[20] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in ACM SIGCOMM, 1989, pp. 1–12.

[21] Y. Zhou and H. Sethu, "On the relationship between absolute and relative fairness bounds," IEEE Comm. Letters, vol. 6, no. 1, pp. 37–39, Jan. 2002.

[22] A. Kumar and J. Kleinberg, "Fairness measures for resource allocation," in 41st Annual Symposium on Foundation of Computer Science. Nov. 2000, pp. 75–85, IEEE.

[23] S. Keshav, An engineering approach to computer networking:ATM networks, the Internet, and the telephone network, Addison-Wesley, Massachusetts, 1997.

[24] A. K. Parekh, A generalized processor sharing approach to flow control in integrated services networks, Ph.D. thesis, Massachusetts Institute of Technology, Feb. 1992.

9 Appendix A: Class Description

9.1 Program Execution Overview

- Packet records are generated according to Bernoulli's process and written into a file. The records are then read from the initial file, randomized and written into another file.
- These records are read from the later file and written into memory (into the main queue) as a linked list of records. This list serves as a queue that holds packet and the order of packets within the queue indicates their order of arrival.
- The profiler classifies the packet in the main queue and places the packet records in the queue designated for the profile. The profiled queue is a linked list of pointers that point to the actual packet record in the main queue. A packet record will not be removed from the main queue until the packet is transmitted.
- The execution of the scheduler is determined by one factor: the number of queues the Egress unit is allowed to hold. The scheduler enters a waiting state when a number of packets that is equal to the number of packets that the scheduler is allowed to hold have been scheduled. The combined length of all the packets within the Egress Queue determines the duration which the scheduler waits before scheduling the next packet. The waiting time is equal to the transmission time of all the packets that reside in the Egress Queue.
- The traffic shaper calculates new bandwidth allocation quotas at the end of the waiting time and the scheduler uses these new values to schedule packets.

9.2 Class Description

9.2.1 BWShape

This class represents the traffic shaper and calculates the adapted percentage values.

Methods:

- *BWShape():* default constructor, which is currently not doing anything
- *BWShape(SimData *setup):* Constructor that initializes the traffic shaper with the current setup data values
- *calcNewPercentageFormula1(int *Flag):* method that performs the calculation of the bandwidth quota for each non-empty profile as indicated by *Flag*. *Flag* is an array whose size is equal to the number of profiles and whose elements can assume a value of 0 or 1 indicating the corresponding profile is empty or non-empty respectively.
- *Round(double in):* method rounds *in* to the nearest long integer
- *update(long length):* method that updates the variable containing the total length of all packets

9.2.2 CSchedulerDlg

- This class represents the core object that coordinates the acquisition of simulation parameters and executes the simulation with the assistance of other classes that will be described later.

Methods:

- *calcBernMulti(int burstID):* This method generates packet records(Length and Profile) for a given burst in which the length of the packets varies according to Bernoulli's distribution. This method will be invoked as many times as the number of bursts as determined by the corresponding simulation parameter. The generated packet records are written into an intermediate file.
- *calcPureRandom():* This method generates packet records with the length of the packets varies according to *random()* function of C++. This function is not designed to generate multi-burst packet records.
- *Clean():* This method frees up allocated memory space used by the queue.
- *factorial(unsigned int num):* This method calculates the factorial of the number passed to it as a parameter.
- *RandomizeBernoulli():* This method reads the intermediate file generated by *calcBernMulti()*, randomizes the packet records within each burst and writes them into a different file.
- *writePackets(Packet *pkt, int Len, Packet lastPkt):* This method is responsible for writing the randomized packets of a burst into an output file.
- *fillQFile():* This method is responsible for reading the packet records from the input file and sets up the input queue to the profiler.
- *init(int schedulerIndex):* This method initializes variables for the scheduling algorithm referenced by *schedulerIndex* (0 for FIFO, 1 for Round Robin, 2 for DRR and 3 for LCWFQ).

- *initSimStatusProgress():* This method sets up the range for the simulation progress bar of the GUI.
- *OnSetup():* This method is responsible for the acquisition of simulation data including the creation of data entry GUIs.
- *OnSimulate():* This method coordinates the overall simulation process after the simulation data has been acquired.
- *print(int id, int index):* This method the delay of a packet given the scheduler identification and the profile number.
- *printDRR(), printRR(), printFIFO() and printLC():* This methods are responsible to update the simulation result for the respective field.
- *refreshDlg():* This method is responsible for updating the current adapted percentage in use every time a new percentage is calculated by the traffic shaper.
- *updateMax(int scheme, int prof, double delay),updateAvg(int scheme, int prof, double delay):* This methods are updates the maximum and average delay respectively of the profile of the current packet, identified by *prof,* since simulation of the current scheduling algorithm identified by *scheme* has begun.
- *updateSimProgress(int Length), updateSimProgress(int Length, int Flag):* These overloaded methods are used to update the simulation progress bar on the GUI.

9.2.3 Queue

This class is designed as a linked list data structure that can represent any queue. It contains a pointer to the master queue used by the profiler and the profiled queues used by the scheduler and the traffic shaper.

Methods:

- *fillQD(int &ref):* This packet is responsible to fill all the profiled queues.
- *getNumPackets(int prof):* This method is responsible the total number of packets that have the same profile as *prof.*
- *getSize(void):* This method returns the total number of packets of all the different profiles.
- *getSize(int profile):* This method calculates and returns the total number of packet records that have the same profile as *profile.*
- *init(SimData *setup):* initializes the queue with the simulation data.
- *insert(Node *newNode):* Inserts a new node at the end of the queue.
- *insertMainQ(Packet pkt):* Inserts a new packet at the end of the main queue.
- *isEmpty():* checks if the queue is empty
- *isProfileEmpty(int prof):* checks if the profiled queue, identified by *prof,* is empty.
- *remove(int &cur, int quantum[], int &ref):* used in the DRR algorithm, it is used to remove the next packet that is due in the current round and updates the quantum value for the specific profile.
- *remove(int &cur, char n, int &ref):* used in the RR algorithm, it evaluates the current profile to be removed and uses the next method to actually remove the packet record.
- *remove(Profile prof, int &ref):* removes the packet record from the profiled queue whose profile is the same as the profile of *prof.*

- *remove(int seqNum, int &ref):* used in the FIFO algorithm, it removes the packet whose sequence number is *seqNum* from the profiled queues.
- *remove(double *credit, int &ref):* used in the LCWFQ scheduling algorithm, it is used to remove the packet whose profile has the lowest credit.
- *removeMainQ():* This method removes a packet at the head of the main queue.

9.2.4 SchedulerSim

This class implements the four scheduling algorithms

Methods:

- *RunFIFO(),RunSRR(),RunDRR(),RunLC() :* These methods represent the algorithms FIFO, Round Robin, Deficit Round Robin and Lowest Credit Weighted Fair Queuing respectively.

9.2.4.1 Setup

This class implements the GUI used to acquire the simulation setup parameters.

9.2.4.2 SimData

This class represents an aggregate data type that is capable of storing all the relevant simulation parameters.

9.2.4.3 QueueLineParam

This class implements the GUI interface that is used to enter simulation data parameters such as line speed, maximum, and minimum silent gaps, maximum and minimum number of packets in each silent gap, scheduler and profile process times, and the number of bursts.

9.2.4.4 QParams

This class represents the data structure that stores the parameters entered through the GUI *QueueLineParam.*

9.2.4.5 Packet

This class represents the data structure that stores the packet record and other relevant information such as sequence number and input time stamp.

9.2.4.6 Profile

This class represents the data structure that stores the information relevant to a profile such as the profile type and the percentage BW quota of the profile.

9.2.4.7 Node

This class represents a linked list data structure that stores an object of type Packet and a pointer to the next Node.

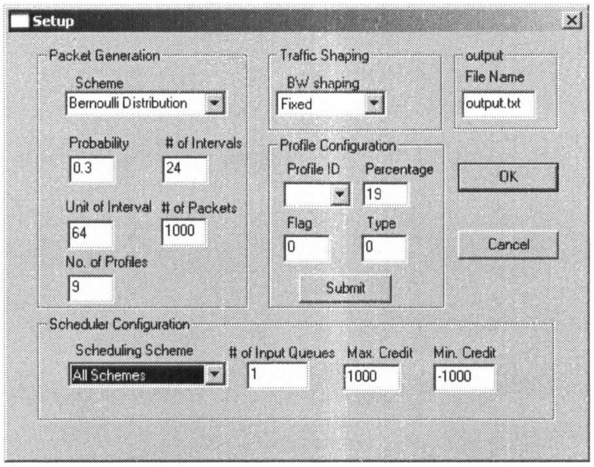
9.2.4.8 MaxAvgFileNameInputDialog

This class is represents the interface that is used to enter the file name to stores the maximum and average delays of all the profiles.

9.2.4.9 DRRQuantum

This class is represents the interface that is used to enter the base quantum value that is used in the simulation of the DRR scheduling algorithm.

10 Appendix B: GUI Screen Shots



Setup

Packet Generation

Scheme: Bernoulli Distribution

Probability: 0.3

# of Intervals: 24

Unit of Interval: 64

# of Packets: 1000

No. of Profiles: 9

Traffic Shaping

BW shaping: Fixed

output

File Name: output.txt

Profile Configuration

Profile ID:

Percentage: 19

Flag: 0

Type: 0

Scheduler Configuration

Scheduling Scheme: All Schemes

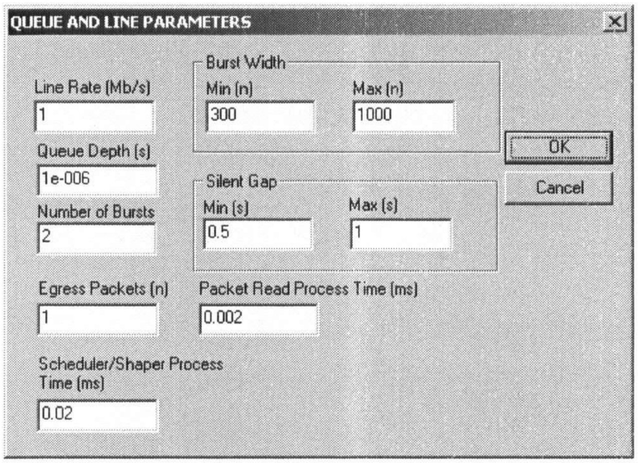
# of Input Queues: 1

Max. Credit: 1000

Min. Credit: -1000

Figure 10.1: Main Setup Screen

Packet Generation is randomized according to Bernoulli distribution or conventional random function. BW Shaping is either Fixed (LCWFQ only with no shaping) or use the adapted Weight Adjustment



QUEUE AND LINE PARAMETERS

Line Rate (Mb/s): 1

Burst Width

Min (n): 300

Max (n): 1000

Queue Depth (s): 1e-006

Number of Bursts: 2

Silent Gap

Min (s): 0.5

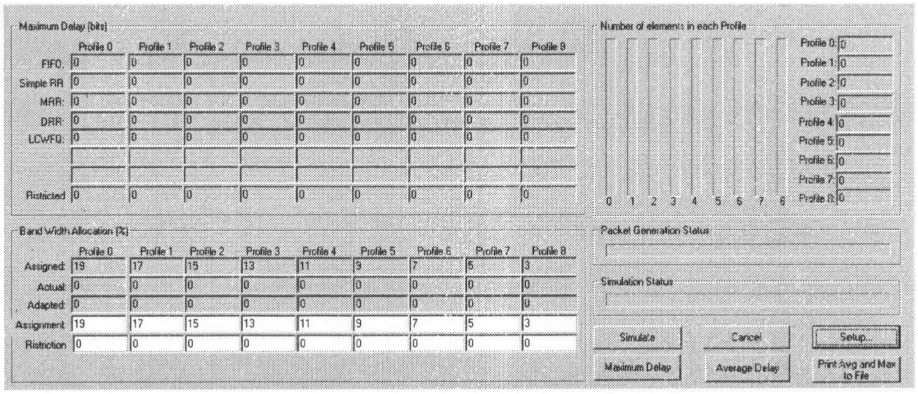
Max (s): 1

Egress Packets (n): 1

Packet Read Process Time (ms): 0.002

Scheduler/Shaper Process Time (ms): 0.02

Figure 10.2: The Queue and Line Parameters



Maximum Delay (bits)

	Profile 0	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 9
FIFO:	0	0	0	0	0	0	0	0	0
Simple RR	0	0	0	0	0	0	0	0	0
MRR:	0	0	0	0	0	0	0	0	0
DRR:	0	0	0	0	0	0	0	0	0
LCWFQ:	0	0	0	0	0	0	0	0	0
Ristricted	0	0	0	0	0	0	0	0	0

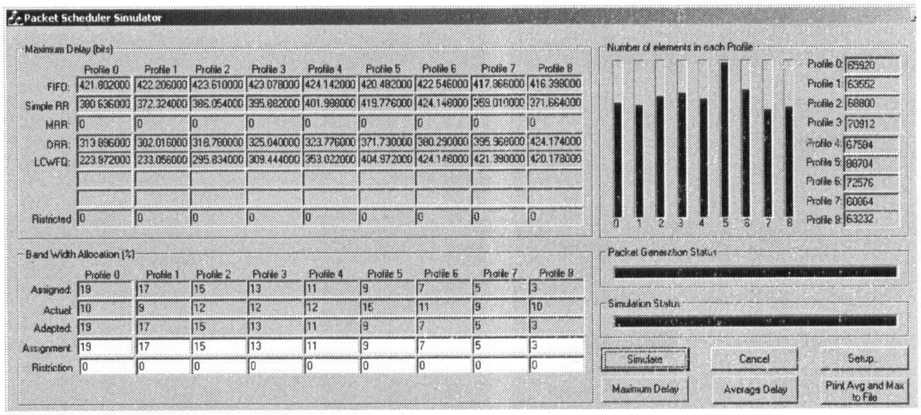
Band Width Allocation (%)

	Profile 0	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 8
Assigned:	19	17	15	13	11	9	7	5	3
Actual:	0	0	0	0	0	0	0	0	0
Adapted:	0	0	0	0	0	0	0	0	0
Assignment:	19	17	15	13	11	9	7	5	3
Ristriction	0	0	0	0	0	0	0	0	0

Number of elements in each Profile

Profile 0:	0
Profile 1:	0
Profile 2:	0
Profile 3:	0
Profile 4:	0
Profile 5:	0
Profile 6:	0
Profile 7:	0
Profile 8:	0

Figure 10.3: Simulation in initial mode before setup



Packet Scheduler Simulator

Maximum Delay (bits)

	Profile 0	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 8
FIFO:	421.802000	422.206000	423.610000	423.078000	424.142000	420.482000	422.546000	417.866000	416.398000
Simple RR	380.636000	372.324000	386.054000	395.882000	401.988000	419.776000	424.148000	359.010000	371.664000
MRR:	0	0	0	0	0	0	0	0	0
DRR:	313.896000	302.016000	318.780000	325.040000	323.776000	371.730000	380.290000	395.968000	424.174000
LCWFQ:	223.972000	233.056000	295.834000	309.444000	353.022000	404.972000	424.148000	421.390000	420.178000
Ristricted	0	0	0	0	0	0	0	0	0

Band Width Allocation (%)

	Profile 0	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 8
Assigned:	19	17	15	13	11	9	7	5	3
Actual:	10	9	12	12	12	16	11	9	10
Adapted:	19	17	15	13	11	9	7	5	3
Assignment:	19	17	15	13	11	9	7	5	3
Ristriction	0	0	0	0	0	0	0	0	0

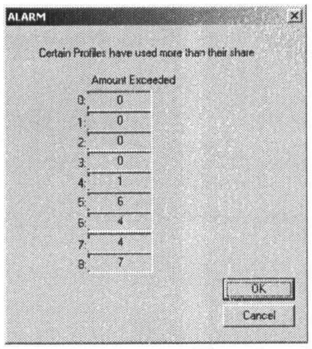
Number of elements in each Profile

Profile 0:	65920
Profile 1:	63552
Profile 2:	68800
Profile 3:	70912
Profile 4:	67584
Profile 5:	88704
Profile 6:	72576
Profile 7:	60064
Profile 8:	63232

Figure 10.4: Simulation completed shows the average delay and maximum delay

The simulator shows:

- 9 profiles with their assigned bandwidth, actual bandwidth and adapted bandwidth
- Packet generated and their distribution
- Maximum delay for the four schemes FIFO, Simple RR, DRR, LCWFQ



ALARM

Certain Profiles have used more than their share

	Amount Exceeded
0:	0
1:	0
2:	0
3:	0
4:	1
5:	6
6:	4
7:	4
8:	7

Figure 10.5: The simulator reports on profiles that have more than their shares