ENHANCING QUALITY OF SERVICE IN INTERNET USING DYNAMIC SCHEDULING

By

ANIMESH DALAKOTI

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of ANIMESH DALAKOTI find it satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

ACKNOWLEDGEMENT

I would like to sincerely thank all those people who have guided and supported me throughout my Master's study at WSU by investing their time, effort, expertise and perspective throughout the work. First and foremost I would like to thank Dr. Behrooz A. Shirazi, my advisor, who not only extended financial support for my graduate study but also provided me a great opportunity to work on this project. He was always there, throughout the research work with his indispensable guidance, profound thoughts and stimulating suggestions. I would like to acknowledge Dr. Murali Medidi, my co-advisor, who helped me in developing fundamentals and later understanding the intricacies of the subject. I would also like to thank Dr. Sirisha Medidi, my committee member, for her continual support and encouragement with her considerate nature and knowledge in the field. I would also like to thank Dr. Sandip Roy whose vital expertise in the field of stochastic modeling helped me in designing the analytical model for the algorithm.

I would also like to thank my colleague Mrs. Nina Marie Peterson, for constantly providing intriguing thoughts and helping me in my research with frequent important conversations and documentations. My fellow graduate students Sunil and Madhusoodan are also sincerely appreciated for their continual support.

And finally, I would like to express sincere gratitude to my parents, my brother Abhishek and my friend Isha who are major sources of my inspiration and have motivated and supported me in all my endeavors.

ENHANCING QUALITY OF SERVICE IN INTERNET USING

DYNAMIC SCHEDULING

Abstract

by Animesh Dalakoti, M.S.
Washington State University
August 2007

Chair: Behrooz A. Shirazi

This thesis elaborates the research work done to achieve better Quality of Service (QoS) in Internet using priority based schedulers. In its early days, the Internet provided Best Effort Services for communication. Later, due to the dramatic surge of Internet traffic stemming from the Internet evolution, classification of traffic flows took place in an attempt to provide a better Quality of Service to particular users based on cost, specifications, and requirements. This thesis introduces a Variable-Weighted Fair Queuing scheduling algorithm (V-WFQ) that provides Quality of Service by dynamically adapting packet priorities to varying network traffic congestion at each router, taking into account the priority of the service being provided. In Variable-Weighted Fair Queuing the changes in congestion at a router will be reflected in a change in the relative priority among flows. V-WFQ provides a prioritization scheme in which higher level Types of Service (ToS) flows dominate the network resources when network resources are constrained, leaving the lower level ToS flows with the remaining resources. This is accomplished through the altering of the flows relative priorities which is implemented with multiple forwarding queues that reflect the type of flow and the congestion level of the network. The industry standard Weighted Fair Queuing (WFQ) algorithm is a

scheduler that uses a static priority mechanism with a predetermined number of forwarding queues. V-WFQ, in addition to providing weighted fair queuing, provides the variability which allows the system to dynamically adapt to the current situation of the network. We compare V-WFQ and WFQ using several QoS metrics including delay, packet loss, throughput, and weighted average system (WAS) delay. We also provide analytical modeling for V-WFQ using Markov chain analysis. Our performance results show that when compared to WFQ, V-WFQ enhances the performance of higher priority traffic while providing comparable performance in terms of throughput and packet drop rate for low priority traffic.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1. Quality of Service

Providing *Quality of Service (QoS)* through the Internet has been a significant research area in recent years [1][2][3]. Researchers have introduced several architectures including the Differentiated services architecture or the Diffserv model [4][5][6] and the Integrated services architecture or the Intserv [7] model, which are later discussed in this report. The network QoS can be defined in a variety of ways and include a diverse set of service requirements such as performance, availability, reliability, and security. All these service requirements are important aspects of a comprehensive network QoS service offering.

QoS refers to the capability of a network to provide better service to selected network traffic over various technologies, be it IP, ATM, Ethernet, 802.11 or a combination of these protocols. Quality of Service can be defined in a qualitative manner that is a relative measure or in a quantitative manner that is an exact measure. The quantitative measure uses absolute network metrics that includes delay, loss and throughput either in terms of bounds or calculated values, whereas relative measures provide a comparison between the services being provided to different classes. The primary goal of QoS is to provide priority including dedicated bandwidth, controlled latency and reduced packet loss across the network. Providing network QoS requires *guaranteeing* resources to the users instead of providing *best-effort* network services [8][9]. The exact form of performance guarantee is a part of the *service level agreement* (SLA) between the network service

provider and its customers [9]. Quality of Service provisioning is suitable for increasing the efficiency of any network, ranging from small corporate to large Internet service providers.

To provide a preferential service to a packet, it must be first identified, this is termed as *QoS identification and marking.* In the current Internet scenario, several service classes simultaneously coexist. The top-most level includes the emergency traffic which is of utmost importance, thus demanding guaranteed QoS. Another service class will provide predictable and satisfying Internet services for companies that provide premium services over the Internet. These companies are the ones providing, faster services to their users, for e.g. Voice over IP (VOIP), webcast, online movies and high bandwidth services requiring fast data transmission. Such companies are willing to pay a certain price to make their services reliable. Then there are the best effort services which do not have urgency for transmission or their owners are unwilling to pay for them. Packet marking or storing the traffic class to which a packet belongs to, can be done in the packet header itself at the source or ingress router [8][9]. Later on, in the network, the class listed in the packet header can be used to provide preferential treatment to the packet or flow.

1.2. QoS controlling parameters

QoS provisions can be achieved through the functioning of several network entities which control the buffer and bandwidth allotment to competing traffic flows. Congestion management, queue management, link efficiency, and shaping or policing tools provide QoS within a single network element [10][11].

a) Congestion Management:

One of the parameter that can control the QoS provided to the traffic is Congestion management. Because of the bursty nature of Internet traffic, sometimes the amount of traffic exceeds the speed of a link. At this point, the congestion control mechanism is activated. The packets get queued at the routers, for future processing. Now QoS can be implemented by controlling the way that the queues are processed. The congestion control mechanism can buffer traffic in a single queue and let the first packet in be the first packet out, or, it can put packets into different queues and service certain queues more often. The class of such congestion control mechanism is generally termed as scheduling algorithms, e.g. First In First Out (FIFO) queuing, Weighted Fair Queuing (WFQ) [12][13], Priority Queuing (PQ) [14] etc, which are the main area of interest in this thesis, and are discussed in detail later in this report [10][11].

b) Queue Management:

When the rate of incoming traffic is greater than the processing rate of the server, the queues are used to store the packets waiting to get processed by the server. But these queues are of finite size and hence they get filled up sooner or later. When a queue is filled, it drops packets from its tail, even if they are high priority packet. Hence for providing QoS we need to make sure that the system processes packets in such a way that high priority queues do not get filled up as much as possible and even if they require to drop a packet they do so for the lower priority traffic.

Congestion avoidance is a form of queue management. It comes into effect before the congestion really develops in to a bottleneck and queues get filled up; this is like unlike

the congestion management algorithms. Congestion avoidance algorithms constantly monitor the network, and provide feedback to the sources to stop the incoming flows when congestion rises, thus preventing occurring of congestion. RED is an example of such mechanism. Random Early Detection (RED) is an active queue management mechanism for congestion avoidance. RED provides enhancement to the primitive FIFO queuing mechanism. The primary drawback with a FIFO queue is monopolization of the network by misbehaving sources. Monopolization occurs when some source tries to capture network resources by sending bulk of traffic to the router [10][11].

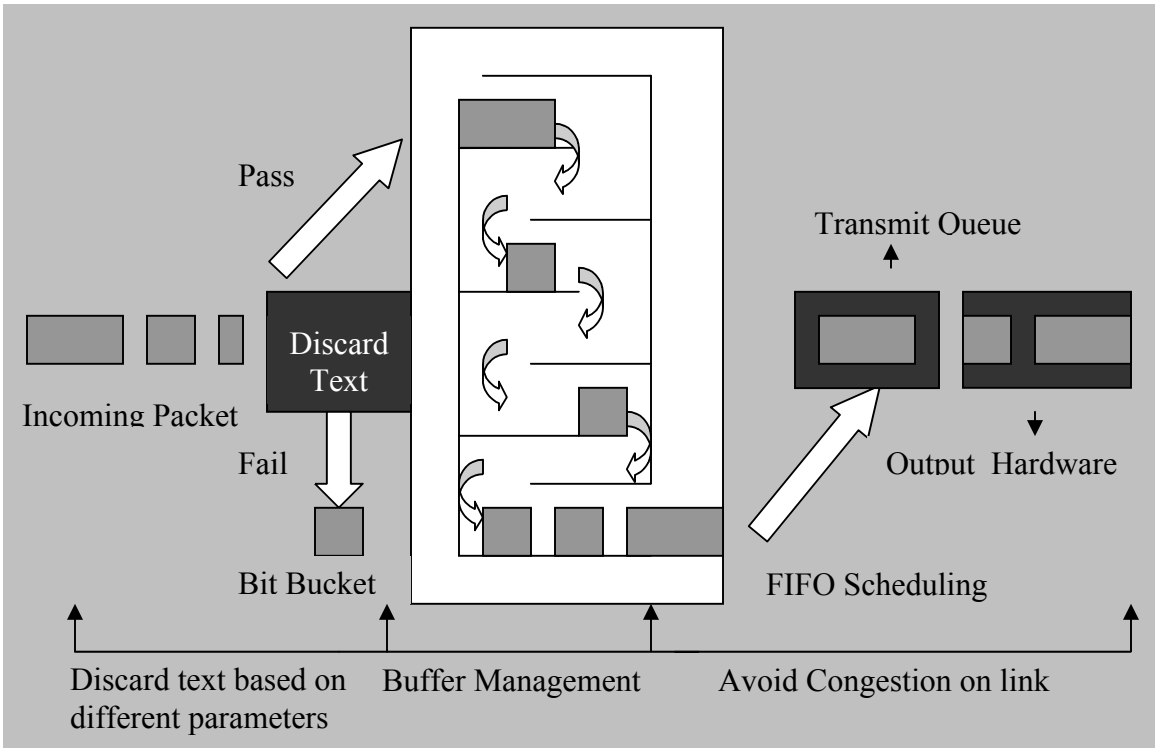

Figure 1: WRED congestion avoidance [15]

As there is a common queue in FIFO, this queue gets filled up and packets of other flows also get affected due to this congestion. This leads to packet drops of even those flows which have slow rate of arrival. Moreover the latency of all flows gets affected when the queue is filled as every packet gets queued up in the same queue. In RED, a

threshold, based on congestion percentage is determines when the system needs to drop a packet. The network is constantly monitored. As the congestion increases the bound, the packet is dropped. The result of the drop is that the source detects the dropped traffic and slows its transmission. Another modification to RED is Weighted RED (WRED) which provides precedence based RED. As shown in Figure 1, WRED provides preferential treatment to higher priority traffics by selectively discarding the packets of lower priority in case of congestion exceeding the threshold level [10][11].

c) Traffic Shaping and Policing:

Shaping is used to create a traffic flow that limits the full bandwidth potential of the flow(s). This is used many times to prevent the overflow problem of the queues mentioned earlier. In shaping, the traffic above the configured rate is buffered for transmission later to maintain the configured rate. *Policing* is similar to shaping, but if the traffic exceeds the configured rate it is not buffered and instead is discarded [10][11].

1.3. Network Scheduling Algorithms

Scheduling algorithms control one way end-to-end delay, throughput, packet loss, and bandwidth management by deciding which packet to forward next from several queues containing traffic from different incoming flows. Providing preferential treatment to a particular flow can provide it with a better QoS. Service guarantees for specific flows are quantitative and can range from simple throughput guarantees to hard bound end-to-end delay and lossless transmissions. Scheduling algorithms are evaluated based on scalability, complexity, and the QoS provisions which provide the requirements

necessary for deployment of the algorithm within the realm of the Internet. The Intserv [7] and Diffserv [4] [5][6] model defined by the Internet Engineering Task Force (IETF)[16] is designed to provide service differentiation among the Internet traffic in order to provide QoS based on classes. Fair Queuing [14] and Weighted Fair Queuing [12][13] are scheduling algorithms being used today in these models. However, these scheduling algorithms lack in performance when the congestion in the system is high. The proposed Variable Weighted Fair Queuing (V-WFQ) is designed to handle such issues and is described later in the report.

CHAPTER TWO

BACKGROUND AND RELATED RESEARCH

2.1 Quality of Service Architecture in Internet

Internet Engineering Task Force (IETF) has defined two models for providing Quality of Service to a given IP packet in the Internet, the Integrated and the Differentiated services architectures. Service differentiation can be broadly categorized as fine grained or coarse grained. In a system providing finer granularity, the network operates on a flow based level, whereas a coarse grained system classifies flows among classes.

A network flow is defined by a 5-tuple, which includes a source IP address, source port number, destination IP address, destination port number, and underlying transmission protocol like a UDP or TCP [16]. A fine grained system is well protected from other misbehaving flows as every flow is allocated separate resources, but due to extra overhead incurred in per-flow operations, the system is non-scalable. The Integrated service architecture is a per-flow based classification, which provides fine granularity among the flows, whereas the differentiated architecture is a coarse grained system, where flows are classified into several classes, each processed differently. The differentiated architecture assumes that all flows within the same class have similar Quality of Service requirements and should be treated similarly.

In order to provide QoS, the flows and their requirements need to be statistically defined. Deterministic bounds on quantities such as loss and delay can be expressed if we combine constraints on traffic flows and service guarantees. The token bucket is the generally used flow specification. A sample flow specification TSpec [16] comprises of a

token bucket with a peak rate *p*, a minimum policed unit *m*, and a maximum datagram size *M*. The parameters *m* and *M a*re used for packet filtering: a packet whose size is less than *m* bytes is counted as *m* bytes and any packet over *M* bytes is considered out of profile. The token bucket has a bucket depth *b*, and a bucket rate *r*, with *b* specifying the maximum burst size and *r* specifying the maximum service rate. When a packet of length *x* is serviced, *x* bytes are removed from the token bucket. If the bucket is empty, the packet must wait in the queue until the bucket fills up with enough tokens. In implementation, a token bucket is often paired with a leaky bucket. Figure 2 demonstrates a leaky bucket for flow specification.

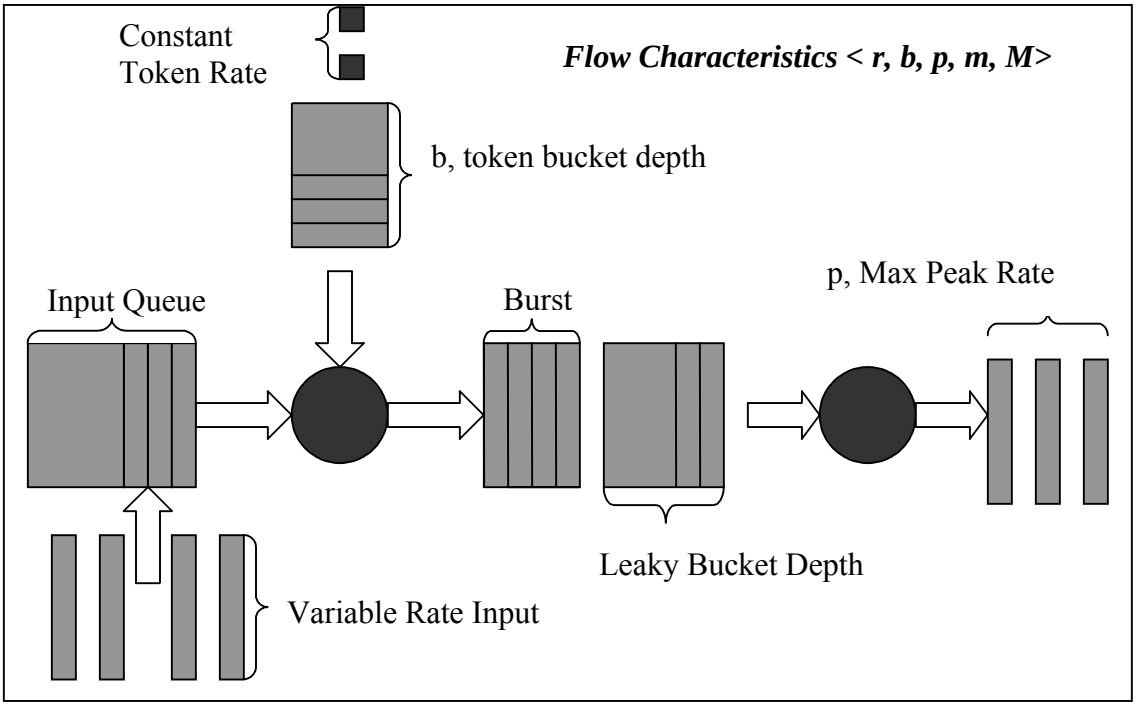

Figure 2: A Leaky bucket for Flow Specification [16]

The specifications of traffic and its desired service can be given on a per-flow basis or in a service level agreement (SLA). An SLA is a service contract between a customer and a service provider, in which the level of service is formally defined. A SLA is a formal

negotiated agreement between two parties. It is a contract that exists between customers and their service provider, or between service providers. It records the common understanding about services, priorities, responsibilities, guarantee, etc. with the main purpose to agree on the level of service. For example, it may specify the levels of availability, serviceability, performance, operation or other attributes of the service like billing and even penalties in the case of violation of the SLA. [9]

Providing end to end Quality of service requires hop by hop data delivery. Hop by hop can also be thought of as data transfer between Autonomous systems (AS). Figure 3 shows an end to end QoS delivery. In today's Internet we have two architectures or framework that provides Quality of Service, Integrated services architecture and Differentiated services architecture [4][5]

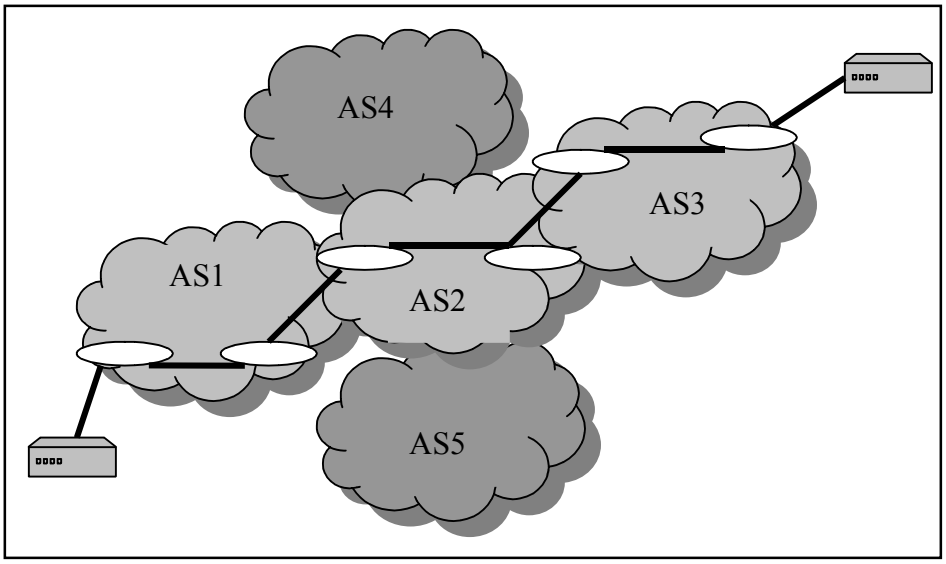

Figure 3: Hop by Hop data transfer between Autonomous Systems (AS) [4]

2.1a. Integrated Services Architecture:

The Integrated services (IS) model is a per-flow based model, i.e. it provides flow level granularity in providing Quality of Service. "A flow can be defined as a

distinguishable stream of related datagrams, which results from a single user activity and requires the same QoS" [7]. An example of a flow can be a web cast transmission between a server and a client. Fundamentally speaking a flow is always defined as traffic between a source and destination pair. Therefore a flow is the finest granularity of packet stream distinguishable by the IS. Integrated services architecture is targeted towards providing two kinds of service to real-time traffic, the guaranteed [8][9] services, for applications that require fixed delay bounds and the predictive service for applications requiring reliable and enhanced services. Guaranteed services provide an upper bound on end-to-end queuing delay. It includes services requiring hard bound delay requirements like the real time applications, emergency situations or may be an administrative data. Predictive services provide a quality of service similar to best-effort service in an underutilized network, with almost no loss and delay [8][9] .

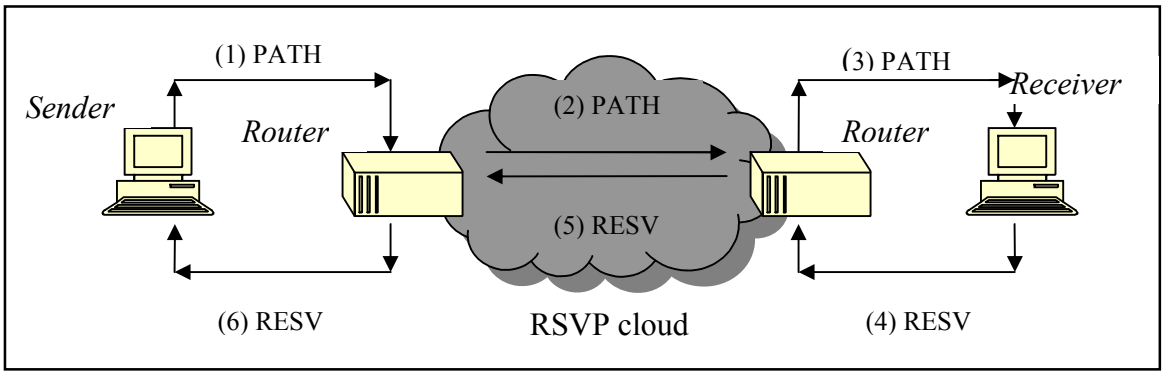

Figure 4: RSVP - The Resource Reservation Protocol [8]

Integrated Services architecture is aimed to share the aggregate bandwidth among multiple traffic streams in a controlled way under overload condition. In an IS architecture the resources must be managed to provide different classes of services. Because all the flows are sharing a common network, IS, needs to reserve resources for

real-time flows to provide better Quality of Service. Hence, resource reservation and admission control is a fundamental step in the IS architecture. IS uses Resource Reservation Protocol (RSVP) as a signaling protocol to explicitly reserve network resources. RSVP is an IETF [16] Internet standard (RFC 2205) protocol for allowing an application to dynamically reserve network bandwidth. RSVP enables applications to request a specific QoS for a data flow, as shown in Figure 4. The sender sends a PATH Message to the receiver specifying the characteristics of the traffic. These characteristics can be any network entity, such as bandwidth. This PATH message gets propagated along the network from source to destination. Every intermediate router along the path forwards the PATH Message to the next hop determined by the underlying routing protocol. When a PATH Message is received at the destination, the receiver responds with a RESV Message to request resources for the flow. Now it is up to the intermediate routers whether to accept or reject the request being made by RESV message [7]. In case the request is rejected by the router the signaling process gets terminated and the receiver is notified about the same. Otherwise the link bandwidth and buffer space are allocated for the flow and the related flow state information is installed in the router. Generally, there are two different possible styles for reservation setup protocols, a connection oriented approach also known as *hard state*, or the connection less approach, also known as the s*oft state*. In a *hard state* approach, the state is created and deleted by the network routers itself. RSVP on the other hand is a soft state reservation protocol, which regards the reservation state as cached information that is installed and periodically refreshed by the end hosts. The state information that becomes stale or is no longer used is timed out

by the routers. If the route changes, the refresh messages automatically install the necessary state along the new route [7].

The Integrated services architecture is implemented by four components. *The resource reservation protocol*, the *admission control routine*, the *classifier* and the *scheduler.*

i) *Packet scheduler:* A packet scheduler schedules the packets waiting to be processed at the router. It is implemented using a set of queues which store the packets temporarily. Its job is to select one packet for processing from several competing packets in different queues. It comes in different flavors, but the versions like priority queuing and weighted fair queuing are most suitable for integrated services architecture.

ii) Classifier: In a priority based system, which classifies the system broadly into some predefined classes, each incoming packet must be mapped into its corresponding class. All the packets belonging to the same class are treated equivalent and are given same level of service by the scheduler. This classification between packets and classes is done by a classifier at the run time. The class to which a packet belongs is embedded in the packet header itself.

iii) *Admission Control:* In order to prevent a new incoming flow from affecting the already guaranteed Quality of Service to some other flows, an admission control policy is established. It implements a decision algorithm which determines whether a new incoming flow can be granted its QoS requirements keeping in mind, all previous flows. Admission control is invoked at each node to make a local accept or reject

decision, at the time a host requests a real-time service along some path through the Internet [7].

Figure 5 shows a typical implementation of all the above mentioned components in the router supporting integrated services. The IS architecture has several drawbacks that prevent it from gaining widespread popularity. The amount of state information increases in proportion to number of flows, which requires huge processing overhead at the routers which render, IS as non-scalable. Moreover all the routers need to implement the RSVP protocol, admission control, packet scheduling etc which is practically not feasible.

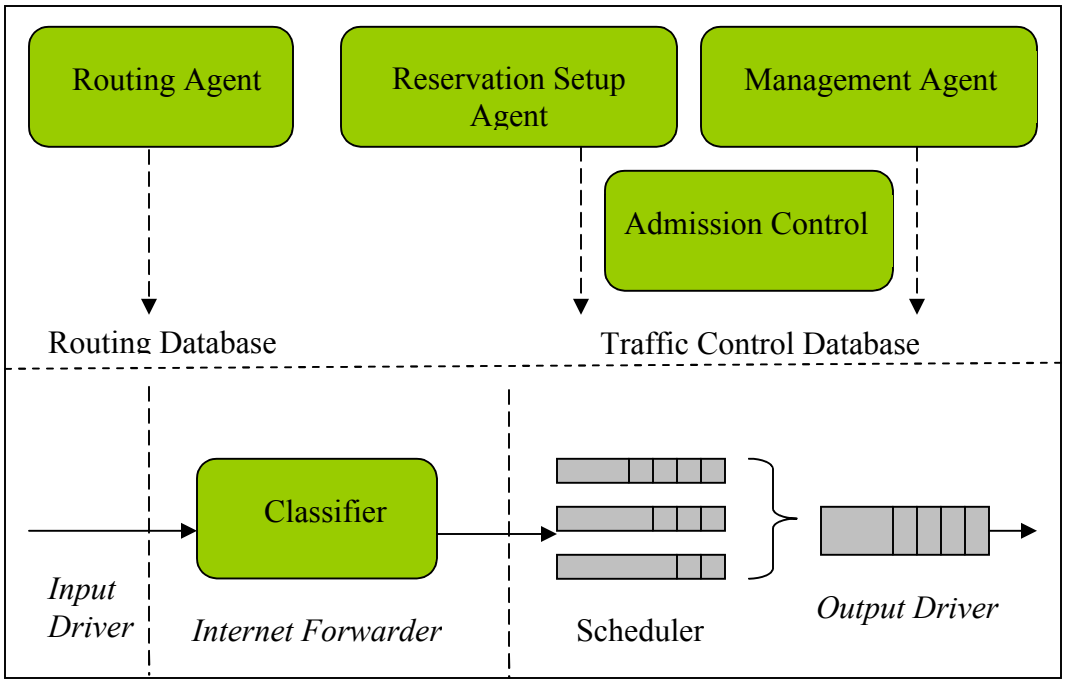

Figure 5: Integrated Services Architecture [7]

***2.1b. Differentiated Services Architecture**:*

The differentiated services (Diffserv) architecture works on the principle of classifying the traffic entering into the network at the network boundaries. The traffic is assigned to different Behavior Aggregates (BA) defined, which is an aggregate

classification of traffic. Diffserv is a framework that provides class based service differentiation and this requires the packets to be marked with appropriate class in the packet header. In IPv4, there is a type-of-service (TOS) byte which is used to mark packets. "The TOS byte consists of a 3-bit precedence field, a 4-bit field indicating requests for minimum delay, maximum throughput, maximum reliability and minimum cost, and one unused bit"[4][5]. These TOS bits have been redefined as Differentiated Service (DS) field that is used to specify appropriate class levels. The six bits of this TOS byte is used to for Differentiated Service Codepoint (DSCP) field whereas the remaining two bits are unused. Each behavior aggregate is identified by a single DS codepoint. For every DS, a Per Hop Behavior (PHB) is defined. It defines the packet forwarding treatment that a DS codepoint packet will get, as compared to others. It is always specified as a relative comparison of the network resources being shared, such as relative weight for sharing bandwidth or relative priority for dropping a packet. The basic architecture of Diffserv consists of following structures as shown in Figure 6 [6].

i) *Differentiated service domain:* A DS domain is a part of network area that is under common service provisioning and consists of set of continuous DS nodes. Every DS is defined with a PHB implemented for every node. The DS is defined by its boundary nodes, whose main function is to classify and condition or police the incoming traffic according the PHB defined in the DS. A typical DS architecture consists of a DS boundary nodes and interior nodes. DS boundary nodes interconnect the DS domain to other DS or non-DS implemented domains, whereas the DS interior nodes only connect to other DS interior or boundary nodes within the same DS

domain. DS boundary nodes act both as a DS ingress node and as a DS egress node for different directions of traffic. Traffic enters a DS domain at a DS ingress node and leaves a DS domain at a DS egress node. A DS ingress node is responsible for ensuring that the traffic entering the DS domain conforms to any agreement between it and the other domain to which the ingress node is connected [6].

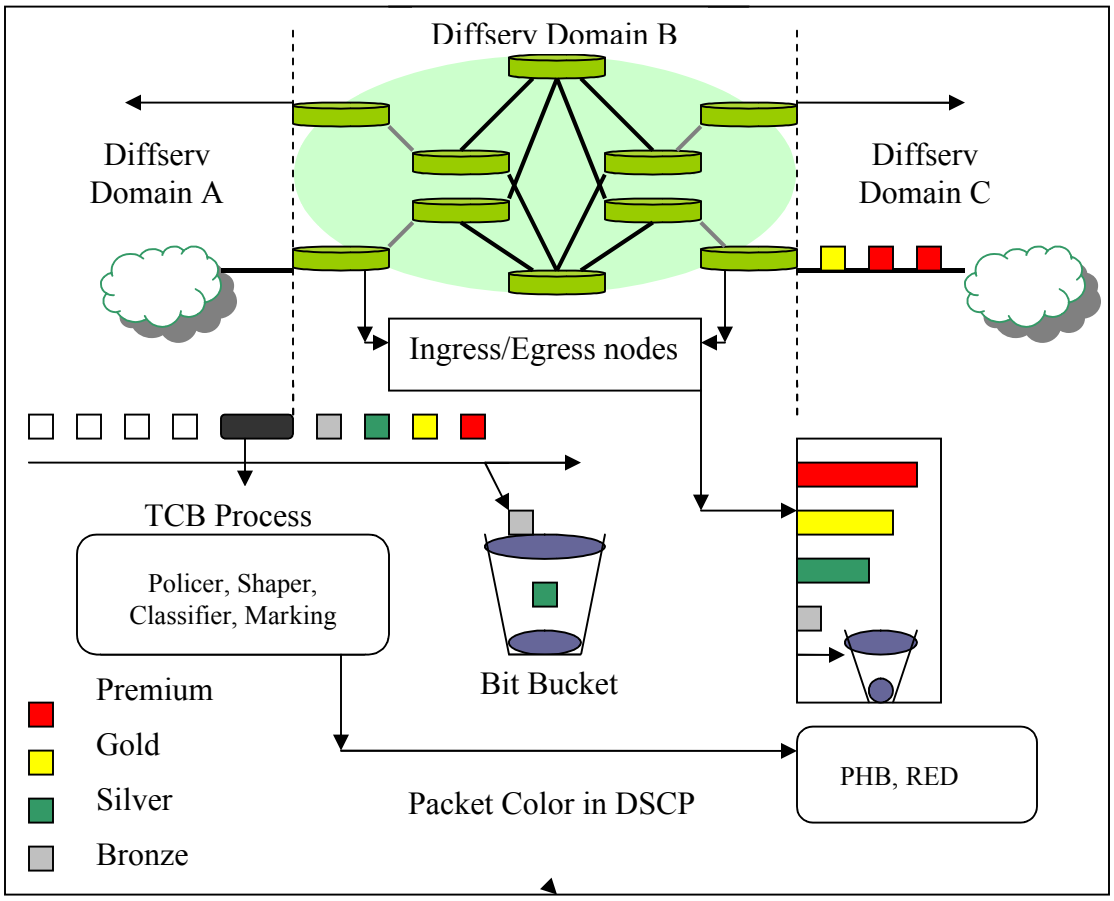

Figure 6: Differentiated Services Architecture [4][5]

ii) *Traffic classification and conditioning:* A service level agreement (SLA) is established between the different Differentiated Services, to extend the architecture globally. The SLA specifies packet classification and re-marking rules and may also specify the policing and shaping actions for the packets which adhere to the profiles defined or disobey them. The packet classification policy identifies the subset of

traffic which may receive a differentiated service by being conditioned and mapped to one or more behavior aggregates within the DS domain. Traffic conditioning performs metering, shaping, policing and re-marking to ensure that the traffic entering the DS domain conforms to the rules specified in the SLA. The traffic conditioning in Differentiated Services architecture is described in detail in Figure 7.

iii) *Classifier:* Its function is to classify the incoming packet into its corresponding class based on the packet header information it carries. The information can be a DS code point, which explicitly specifies its class. The behavior aggregate classifier classifies packets based on the DS codepoint. Hence, classifier diverts the packet matching some specified rule to an element allocated to that class.

iv) *Traffic meters:* It measures the characteristics of the stream of packets to compare them with the traffic profiles specified in the SLA. The information collected by the meter is fed to the other network entities like shapers and droppers which in turn take the required action depending upon whether the traffic is in or out of profile.

v) *Packet markers:* It sets the DS field of a packet to a particular codepoint, hence allocating the packet to a particular DS behavior aggregate defined. The marker may be configured to mark all packets which are steered to it to a single codepoint, or may be configured to mark a packet to one of a set of codepoints used to select a PHB in a PHB group, according to the state of a meter.

vi) *Shapers:* Its function is to shape the incoming traffic to match the traffic profile already specified. It delays some or all of the packets in a traffic stream. The shaper has a finite size buffer and uses this buffer to shape the traffic by dropping the excess packets once the buffer limit is reached [6].

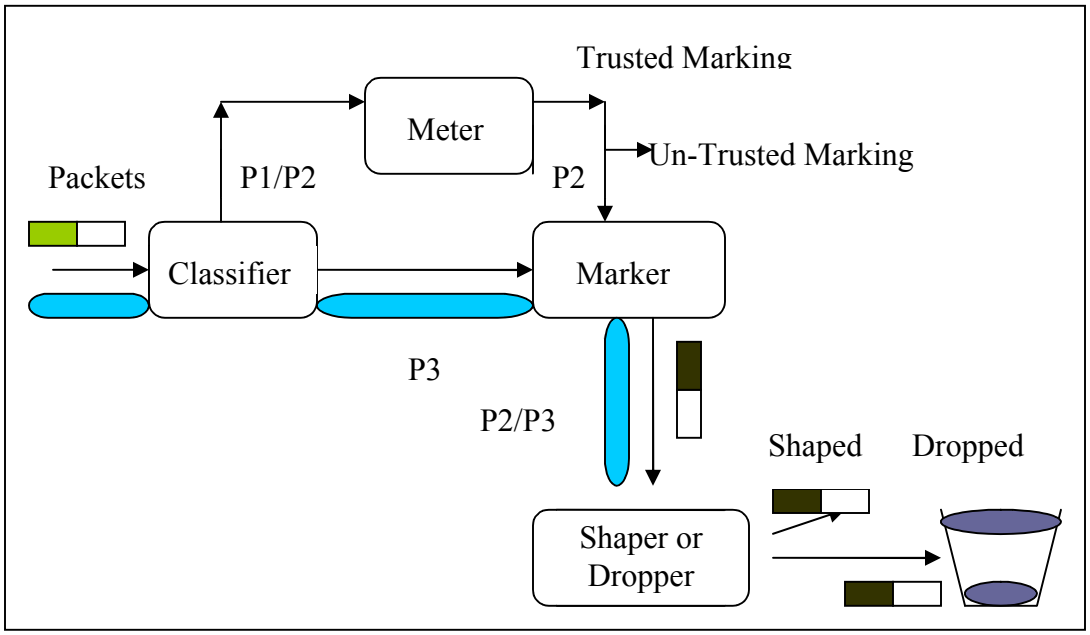

Figure 7: Packet Processing in Differentiated Services Architecture [4][5]

vii) *Droppers:* It discards some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This process is known as *policing* the stream.

Hence, Diffserv has two important design principles, namely pushing complexity to the network boundary and the separation of policy and supporting mechanisms. Since a network boundary has relative small number of flows, it can perform operations at a fine granularity, such as complex packet classification and traffic conditioning.

2.2. Scheduling Algorithms

Packet latency is an important factor that is used in determining Quality of Service. Packet latency in a network is determined by transmission delay, propagation delay, processing delay and queuing delay. All other delays are dependent on physical characteristics of the network except the queuing delay, which is the waiting time that the packet spends in the queue before being transmitted. It is also used to handle the overflow

of arriving traffic; hence it also controls the throughput and packet drop for a flow. A scheduling algorithm basically sorts the packets based on classes and then uses some priority to process these packets on the output link of the router. Our research is mainly focused on the scheduling algorithm, and hence we describe here some of the packet scheduling algorithms that led us to investigate this field and provide an enhancement over the past works.

i) *First In First out scheduling (FIFO):* It is a simple store and forward capable scheduler. In FIFO queuing, the packets are stored in a queue during congestion and are processed in order of their arrival. Although FIFO is a default queue strategy, it has several inherent faults as far as packet forwarding for Quality of Service in networks is concerned. It does handle QoS very well, as it no provision for priority, and processes packets based on their order of arrival. Moreover any misbehaving flow can create trouble for the entire system by sending out bulk of data, leading to the overflow of the FIFO queue, which results in packets of other flows getting dropped. Because of its nature of treating all network traffic with the same importance, FIFO cannot provide quality of service to time sensitive data.

ii) *Prioritized Queuing (PQ):* Priority queuing is a strict priority based scheduling biased towards the higher priority traffic. It can prioritize based on incoming interface, source or destination address, routing protocol etc. A sample PQ scheduler architecture is shown in Figure 8. It maintains a fixed set of incoming queues, for storing in coming packets based on priority levels say high, medium, normal and low. Now in the processing of packets, the scheduler gives absolute priority to higher level packets, that is, if there is any packet present in a higher priority level queue, it will be processed prior

to any other packet in other queues, no matter how long the low priority lows have been waiting in the system. PQ can lead to starvation problems due to its strict priority mechanism. PQ is a static allocation and does not adapt to the changing network environments.

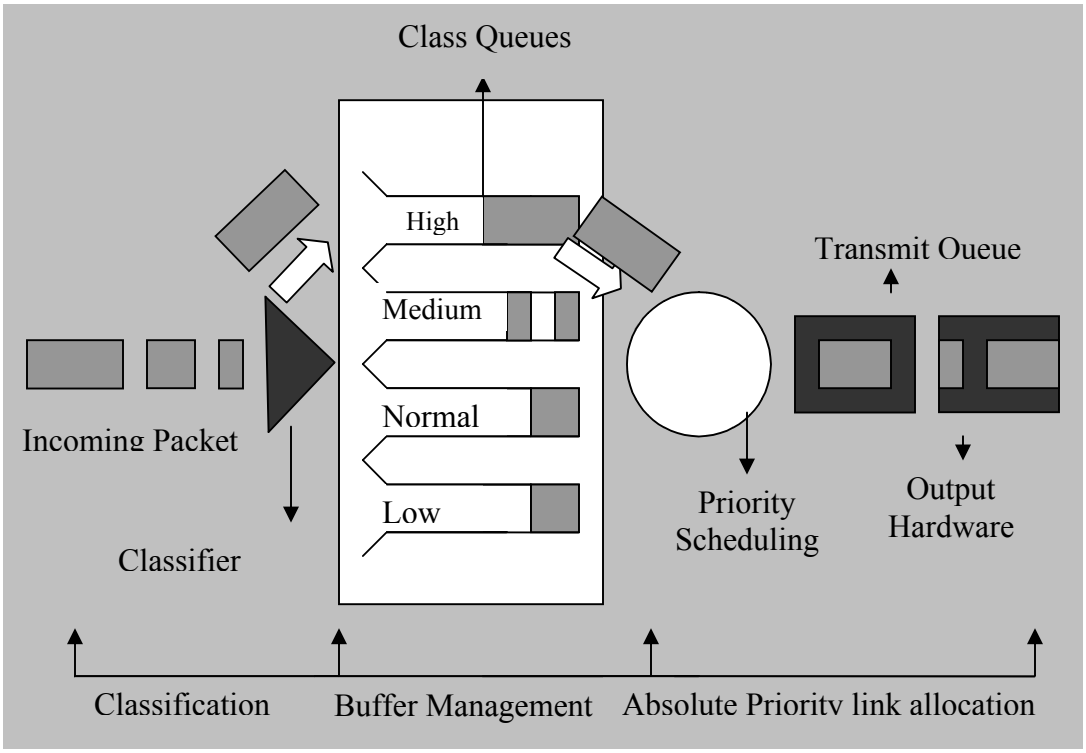

Figure 8: Priority Queuing Scheduler [15]

iii) *Custom Queuing (CQ):* CQ is an extension to the priority based queuing. Instead of providing strict priority among flows, CQ is designed to allocate the network resources among the competing users or applications that have a minimum bandwidth or latency requirements. The bandwidth is proportionally shared among the competing users or applications as shown in Figure 9. Custom queue works by allocating a specified amount of queue space to each class of traffic and then processing them in a round robin fashion. Hence every flow is guaranteed to get at least some fixed amount of service every round unlike strict priority queuing which does not processes lower priority flows if the queues

reflecting higher priority queues are non-empty. In case some queue is empty, its share of bandwidth is fairly shared by the remaining queues.

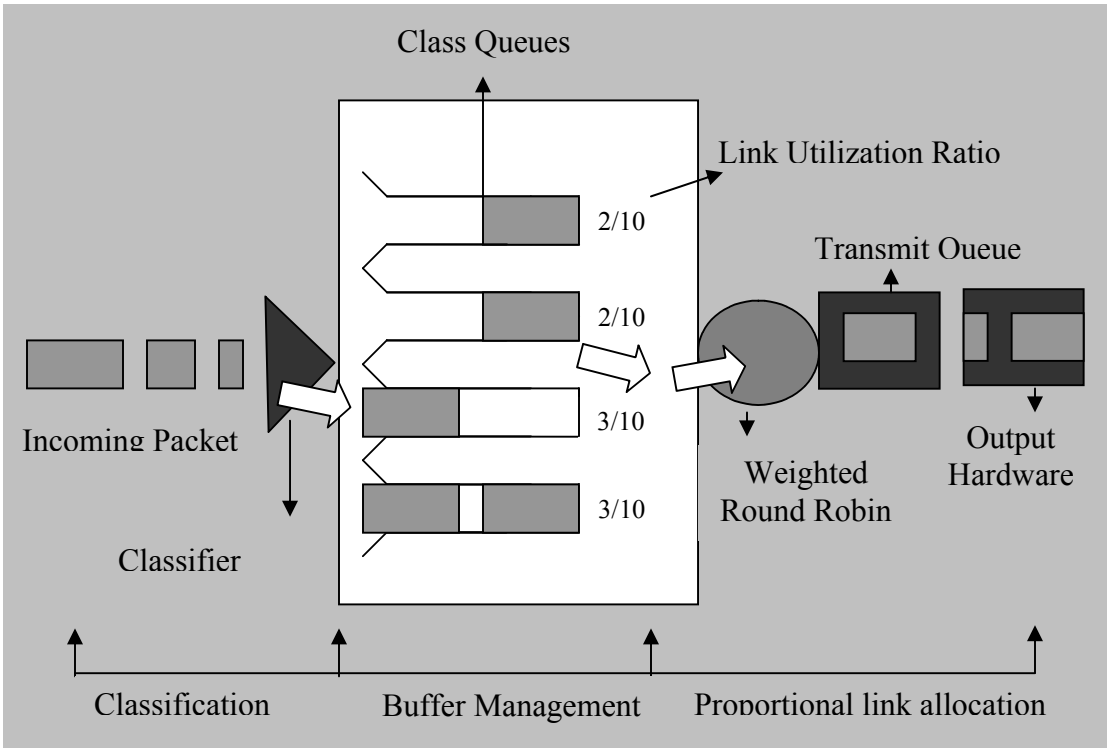

Figure 9: Custom Queuing Scheduler [15]

iv) *Weighted Fair Queuing (WFQ):* WFQ is a packet scheduling technique allowing improved bandwidth services. It is a flow based scheduling algorithm that lets several flows share the same link. WFQ schedules the interactive traffic to reduce response time and also fairly shares the remaining bandwidth among the flows. It is an approximation of Generalized Processor Sharing (GPS) [12][13] that is a generalization of Processor Sharing (PS) algorithm [14]. In PS each session has a separate FIFO queue. At any given time the N active sessions or the backlogged queues (non-empty queues) are serviced simultaneously, each at a rate of $1/N^{th}$ of the link speed. Contrary to PS, GPS allows different sessions to have different service shares, due to different weights assigned to every queue. Since each session has its own queue, a misbehaving source will only

punish itself and not other sessions, unlike the FIFO queuing. The CQ does not work well with packets of different sizes, whereas the WFQ does not make any assumption about the packet size. WFQ ensures that queues do not starve for bandwidth, and that traffic gets predictable service. WFQ is based is on fluid-flow fair queuing and services each queue fairly in terms of byte count creating bit-wise fairness. Class based weighted fair queuing as shown in Figure 10, is an extension to WFQ and is a more practical algorithm for today's Internet. Instead of providing simple flow based model, it extends the system into a scheduler capable of serving user-defined classes. The classes are defined based on one or more of several factors that include protocols, source and destination addresses, and access lists etc. Once a packet is sent from the host, it is associated with a certain class, based on the characteristics defined. A queue is reserved for each class at the scheduler and the packet is queued up on its corresponding class on arrival at the scheduler.

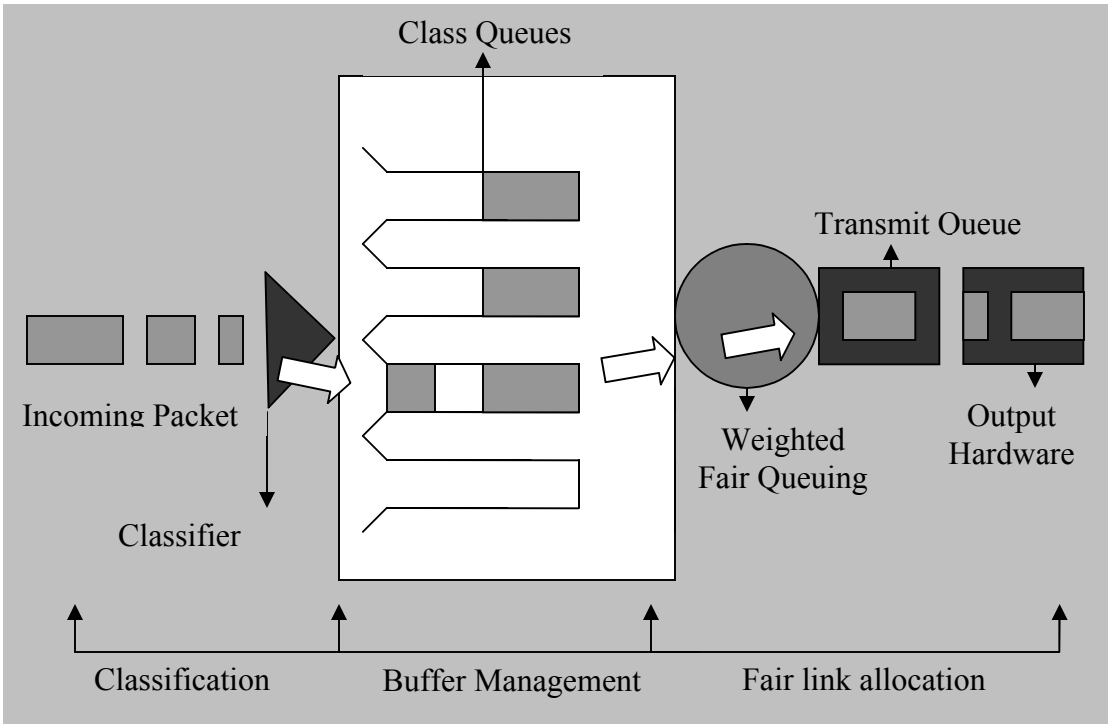

Figure 10: Weighted Fair Queuing Scheduler [15]

In a class based WFQ, When a packet arrives at a router, it is classified and placed in a class-queue depending on the class it belongs to. The weights are assigned to queues, which assign the corresponding packet an equivalent weight, depending on what class of flow it belonged to. The weight of a class is hence user configurable. This class based WFQ provides coarser granularity by working on class level rather then individual flow level.

2.3. WFQ: The Analytical Model

Weighted fair queuing algorithm is an approximation to the fluid reference model, which is shown in Figure 11. The fluid reference system assumes some finite number of incoming flows in the systems, which are being served relatively at the server, based on the queue weights. These flows comprise of packets of some finite size. It serves simultaneously the head of lines packets of all the backlogged queues in infinitesimally small units that are proportional to the specified queue weights [18]. The fluid reference system, working in infinite granularity serves the incoming flows and optimizes the packet queuing delay without violating fairness, as the service among the queues is shared in proportion to the queue weights.

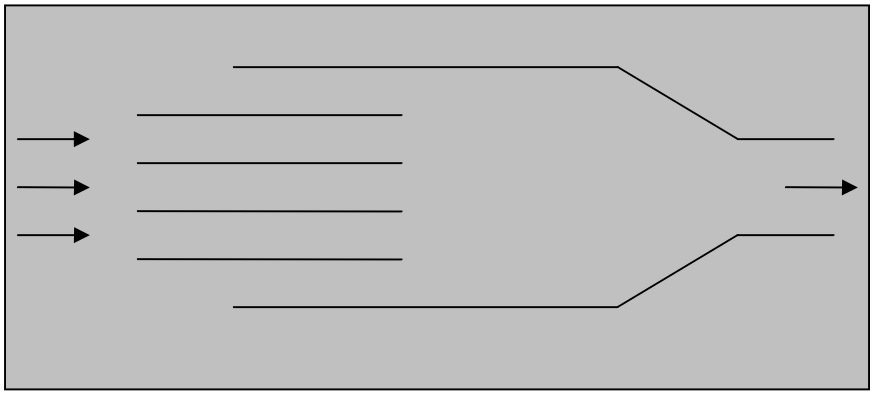

Figure 11: Fluid Reference Model [18]

But, serving packets in infinitesimal size can not be implemented in real world situations as packet exist as a entity in itself, and hence they can be served in entirety or not at all. No such concept exists of processing packet in by subdividing it into smaller sizes. But the WFQ scheduling algorithm, processes the queues on packet basis, and still maintains the processing order of the fluid reference system, i.e. every packet leaves the system in order it would have left in the fluid system. WFQ, implements the packet based version of the fluid reference system by using the departure time tags as shown in Figure 12. These tags reflect when a packet must have departed from the fluid reference system. WFQ assigns all the incoming packets with a start and finish time tags. Based on the finish or departure tags, WFQ sorts the packets in this packet based system [18].

There can be a lag in serving the packets in a packet based system as compared to a fluid reference system as the packets are served as entities not as infinitesimal portions. Hence, it is very much possible that a packet leaves the system later than fluid reference system. The delay introduced in approximating the fluid reference using departure time tags in a packet based system has a maximal bound.

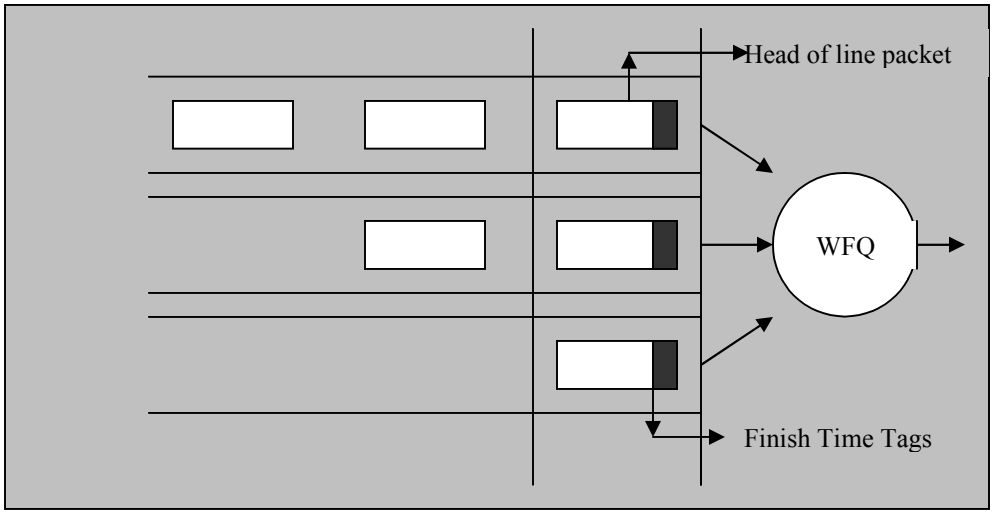

Figure 12: Packet Based System [18]

Parekh and Gallagher[12][13] have shown in their paper that "a packet must have departed in the packet based system no later than the time needed to transmit one maximum-size packet of the packet-based-system after its departure from the fluid system". Using this limit, one can give a tight bound on the maximum delay for a packet in any queue in the WFQ. So, the underlying theory is, the scheduler processes the packets from the head of the queues based on the smallest finish time tags. But the question is how to calculate them in the packet based system. The problem is, the departure time, depends on the service rate of the queues.

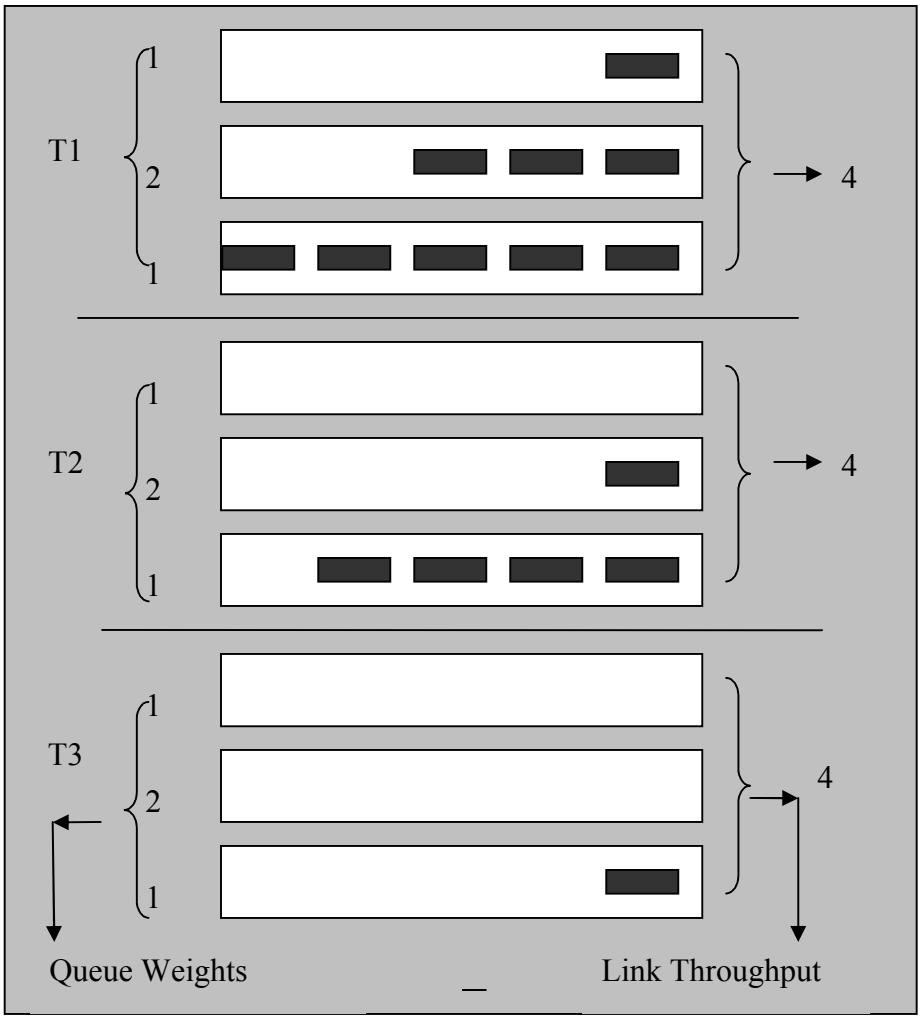

Figure 13: Sample processing by WFQ [18]

The service rate itself depends on the number of backlogged queues in the system, as the network resources like the bandwidth gets distributed in the proportion of the weights of these queues. The empty queues don't count in this distribution. Now the number of backlogged queues changes with time and hence the relative service rates as shown in Figure 13. Therefore departure tags calculated earlier in the system will become voids once the number of backlogged queues changes. Re-computing departure tags of every packet in the queues, every time the set of backlogged queues changes is an expensive task. Parekh and Gallagher [12][13] introduced the notion of *virtual time* to keep track of the arrival and departure times in the fluid based system. *Virtual time* nullifies the effect of change in set of backlogged queues by modifying the concept of time. In virtual time frame all the queues are assumed to be backlogged at every instant.

Let say that for an N queue system, the queue weights are Φ_1, Φ_2 …Φ_N with every weight > 0. The overall link rate at the server is R. The set of backlogged queue in the system is B. This set changes with time as the number of packets in the queues changes from 0 to some positive value. The service rates of every queue also changes with the change in set B, as the link rate R gets distributed among the backlogged queues in proportion to the weights assigned to the *n* backlogged queues. Let us say that during time interval (δ1, δ2] the set B of backlogged queues remains constant and the server serves all *n* backlogged queues simultaneously The service rate allocated to some queue *n* in this time interval is given by equation (1). It is the guaranteed service rate allocated to the queues. If there is any excess bandwidth available, due to some empty queues it is fairly shared between the backlogged queues in proportion to their weights Φ_n.

$$r_n(t) = \frac{\Phi_n}{\sum j \in B(\delta 2)\Phi_j} \times R, t \in (\delta 1, \delta 2] \qquad (1).$$

Parekh and Gallagher [12][13], introduced the virtual time system, in which the packets are always served at a constant service rate $\Phi_n \times R$, unaffected by the arrival and departure of packets in the system which effects the number of backlogged queues. Hence, the change in the service rates due to change in B is nullified in the virtual system. The virtual time V(t), indicates how the weights gets transformed into the service rates received by the backlogged queues at some time t. V(t) is defined to be 0 for idle fluid reference system.

Let us assume, the set of backlogged queues don't change in time interval *dt.* The service rate received by the backlogged queues in real time in time *dt* is:

$$r_n(t).dt = (\Phi_n \times R) \times (V(t+dt) - V(t)) \qquad (2).$$

Equation (2) reflects that the service received in real time interval *dt* with service rate $r_n(t)$, is same as service received with rate $\Phi_n \times R$ in virtual time V(t +dt)- V(t) So, *virtual time* can be defined as,

$$V(t+dt) = V(t) + \frac{dt}{\sum j \in B(t) \Phi_j} \qquad (3).$$

Using current allocation of service rates from equation (1), one can calculate the start and finish time tags for arriving packets. The start time tag *S* reflects the time when the processing of the packet would have started in a fluid reference system. The finish time tag *F* reflects the time when the packet would have departed in a fluid reference system after being processed. The start and finish time tags associated with the packets wont change in case the set B of backlogged queues change, as in this virtual world the queues are always processed with a constant service rate. The start and finish time tags indicate the worst case start and departure time tags, i.e. when all the queues in the system are

backlogged and hence the queues receive same service rate both in the real and virtual world. Now, as mentioned earlier the packets are processed in order of these finish time tags *F,* which results in same order of processing as in fluid reference system, but may be delayed in time as compared to fluid reference system.

The start and finish time for a k^{th} packet in queue *n* at time $a_{n,k}$ having length l_n^k can be calculated as,

$$F\, n^0 = 0 \qquad (4).$$

$$S_n^k = max\,(F_n^{k-1},\ V\,(a^{-1}_{n,\,k})) \qquad (5).$$

$$F_n^k = S_n^k + l_n^k / r_n \qquad (6).$$

where $a^{-1}_{n,\,k}$ is the time just prior to the arrival of current packet. The *max* operation is used to check the condition where the queue *n* is empty when packet *k* arrives although it has already received service since the last idle period. Hence, once the finish time tag *F* has been calculated, the scheduler looks for the smallest of them among the head of line packets in all queues and then calculates the real time equivalent of the virtual finish time for the same. The current virtual time and queue service rate are used to convert virtual time to real time. Let say, F_{min} is the smallest finish tag in the system; its corresponding real world finish time can be calculated from the equation (7) [14][15].

$$F\,min - V(t-) = (next(t) - t) \times \frac{R}{\sum j \in B(t+) rj} \qquad (7).$$

$$next(t) = t + (F\,min - V(t-)) \times \left(\sum j \in B(t+) rj\right) \Big/ R \qquad (8).$$

Using *virtual time*, there is no need to recalculate arrival and departure time for every head of line packet in the queues on packet arrival and departure even if the number of backlogged queues changes in the system.

2.4. Network Simulation Tools

Communication network are complex systems, therefore in order to simulate the networks algorithms, simulation environments are generally used. We have used the NS network simulator to implement and compare our algorithm. NS is an object oriented event driven simulator written in C++, with OTcl interpreter as a front end. Ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. It supports two kinds of class hierarchy, a compiled C++ hierarchy and an interpreted OTcl hierarchy. The two hierarchies are closely related to each other and have one to one correspondence between each other. The root of the hierarchy is the TclObject. The new objects are created using interpreter, which instantiates them and are mirrored by a corresponding object in compiled hierarchy.

The data generated from the simulation results were stored in text files, which were later parsed using the PERL script to produce meaningful interpretation from the data.

CHAPTER THREE

VARIABLE WEIGHTED FAIR QUEUING – VWFQ

3.1. Overview

Our proposed congestion-reactive V-WFQ algorithm has been designed to assist the Diffserv [4][5] architecture in providing Quality of Service performance improvements for high priority messages. We assume the packets have four priority levels based on their type of service consistent with the Diffserv architecture: Emergency services, Paid Premium services, Administrative services, and General services [8][9]. These services are processed in order of their priority within Diffserv architecture. Thus, Emergency services have the highest priority while General services have the lowest priority. V-WFQ provides an enhancement over the WFQ algorithm by providing better QoS to urgent traffic even in the case of increased congestion.

WFQ is a static scheduling algorithm, in which every flow/queue is associated with a fixed weight and packets are scheduled based upon the ToS-level requirements. However, these static scheduling algorithms are unable to adapt to changes in congestion at each router, making them incapable of handling run time changes in the priorities of the flows. Our proposed V-WFQ provides a congestion-reactive Variable-Weighted Fair Queuing algorithm which is a dynamic QoS provisioning scheduling algorithm, allowing it to be deployed within the current Diffserv architecture with some modifications. Unlike WFQ, which has a one to one static mapping, V-WFQ has a dynamic mapping that changes with current congestion, and is implemented through a Priority Lookup Table (PLT).

Let *nQueue* be the number of queues at the scheduler and *nToS* is the number of ToS levels. In WFQ, *nQueue* is always equal to *nToS* which is a direct mapping, shown in Figure 14.

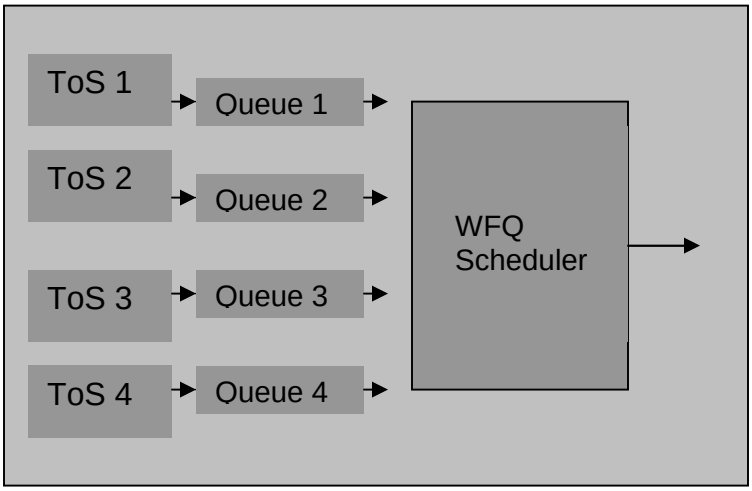

Figure 14: WFQ nQueue, nToS direct mapping

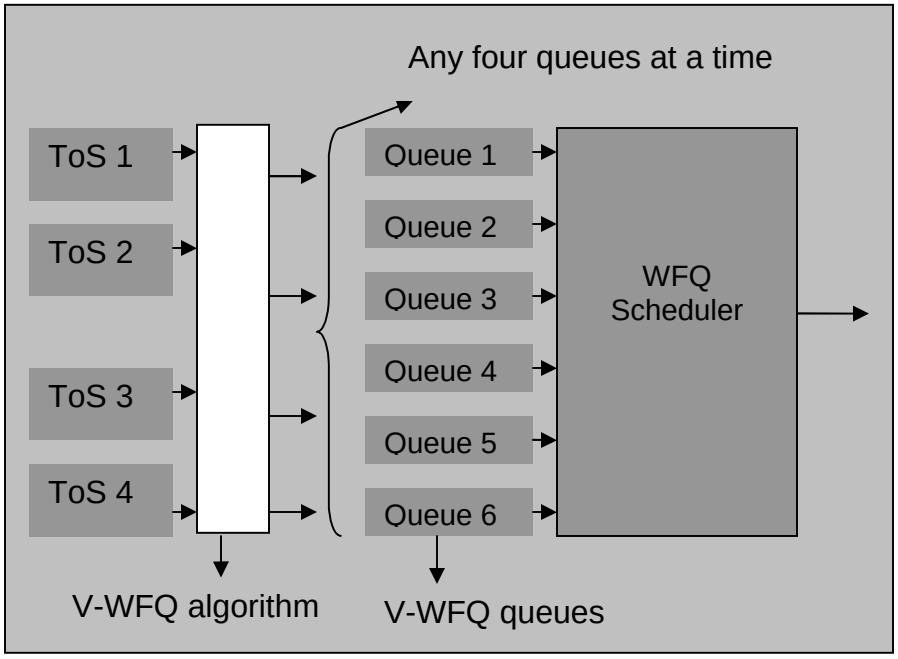

Figure 15: V-WFQ nQueue, nToS associative mapping

However, in V-WFQ, *nQueue* is always greater than *nToS* providing as associative mapping as shown in Figure 15. The additional queues in V-WFQ are used to push low priority messages into lower priority queues when the network congestion levels are high. The idea is that high priority messages remaining in the higher priority queues will have a better chance of service compared to the low priority messages. The algorithm is implemented using an 8-bit field in the message header called P-bits (for priority bits). For a given message, its P-bits value (or priority level) will dynamically change as a packet is forwarded from one router to the next according to the levels of network congestion.

3.2. Design Goals

The Variable Weighted Fair Queuing was designed to tackle the inability of WFQ to handle the change in priorities among different levels of traffic with the change in congestion in the network. The queuing delay experienced by a packet in the network is dependent on the congestion in the network which in turn is determined by the queue lengths. In WFQ as the congestion increases, the delay experienced, even by the higher priority packets, increases because of increase in queuing delay. What we want to achieve is to increase the relative priority of higher priority packets as the congestion increases so that the effect of increase in congestion is nullified for the higher priority packets and they still receive their guaranteed service. The algorithm provides preferential service to the higher priority traffic by stealing the network resources from the lower priority traffic as the congestion increases. During periods of low network congestion, the algorithm reduces to WFQ algorithm with the basic set of weights.

3.3 Protocol Specification

The V-WFQ algorithm proposed in this thesis is an enhancement over WFQ algorithm. Hence it is build over the underlying WFQ algorithm with several other additional modules working over it. The protocol is built on other entities like the PLT table and Shifter which are defined in the following section.

3.3a Initialization Phase

When a packet is sent out from a source, it's P-bits (8-bit field) in the message header is set to all 1's. An all 1's P-bits value indicates a situation in which there is no or low levels of congestion on the communication link. When the P-bits are set to all 1's, V-WFQ behaves the same as WFQ in scheduling a router's incoming queues. When a packet arrives at a router, its P-bits will be changed according to the congestion level on the router's output link. Thus, through the manipulation of the P-bits we will keep higher priority packets in higher priority queues while pushing lower priority packets to even lower priority queues as explained below.

Table 1: Defining ToS and level of services

TOS	Definition	ToS bits
Level 1	Emergency Services	111
Level 2	Administrative Services	110
Level 3	Paid Premium Services	101
Level 4	General Services	100

When the message is put on the output link of a router, its P-bits will be reset to all 1's again. Thus, P-bits will dynamically change as a packet moves from source to destination, effectively changing the priority of the packet according to the congestion levels at each router. The priority of a packet is set by the service application using ToS-bits (3 bits for Type of Service). Table 1 displays a description of each of the four Types of Service as well as their corresponding ToS-bits in decreasing order of importance.

3.3b. The Architecture

The system architecture consists of four modules: the Shifter, the Scheduler, the Priority Look-up Table (PLT), and the Classifier. The Shifter and PLT are not used in the current implementation of the QoS model [4][5] explained above and are introduced as new modules within the architecture for the implementation of V-WFQ. Table 2 shows a sample PLT. The PLT is essential to V-WFQ as it is responsible for determining how the shifter shifts the P-bits whenever a new packet arrives at a router. In WFQ there are as many queues as the ToS levels. Thus, with 4 ToS levels, there will be 4 queues and the ToS value will determine which queue a packet is assigned to when it arrives at a router. In V-WFQ, however, more input queues are utilized at each router (number of queues are determined by the network administrator). In the PLT, shown in Table 2, it is assumed we have 6 queues for the 4 types of service, labeled as P, P/2, P/4….P/32, with queue P having the highest priority and P/32 having the lowest priority. When a packet arrives at a router, its P-bits value will be right-shifted to lower its priority according to the congestion situation on the output link of the router. For example, when a packet of ToS = 111 arrives and the congestion level is at 85%, then its P-bits is right-shifted twice and

the packet is assigned to the P/4 queue. On the other hand, if the congestion level is at 65%, the packet will be right-shifted once and assigned to the P/2 queue.

Table 2: Priority Look-Up Table (PLT)

ToS	% Congestion at Router X	Input Queues at Router X
Level 1 Emergency ToS – 111	45 65 85 100	P P P P/2
Level 2 Administrative ToS – 110	45 65 85 100	P/2 P/2 P/2 P/4
Level 3 Paid Premium ToS – 101	45 65 85 100	P/4 P/8 P/8 P/16
Level 4 General ToS – 100	45 65 85 100	P/8 P/16 P/32 P/32

The Priority Look-up Table must be defined by the network administrator. The right-shifted value of the P-bits (according to the PLT) determines the queue to which each packet will be scheduled. The PLT defines the mapping from which the P-bits should be right-shifted based on the current congestion level at the router and the ToS of the packet. The PLT has been designed so that as congestion increases, higher level ToS flows are designated to queues with higher priorities while lower level ToS flows are shifted down to queues with lower relative priorities. This ensures that as congestion increases, the higher level ToS flows are given an increased share of the network resources.

Table 3: Classification Table and Queue priorities in WFQ

PLT entry	Queue No	Weight for WFQ
P	1	25
P2	2	15
P/4	3	9
P/8	4	5
P/16	5	3
P/32	6	1

Table 3, shows an example classification table. This table is used by the classifier to allocate packets based on their new (shifted) P-bit values to the input queues at each router. Once the packets are classified into the input queues, they will be processed based on the WFQ algorithm in which packets are sent from the queues to the router on a round-robin basis. However, queues are weighted such that more packets are processed from the higher priority queues at each turn. In the example of Table 3, first 25 packets are processed from the P queue, then 15 packets from the P/2 queue, and so on. It can be seen from the PLT that as congestion increases higher relative weights are assigned to the higher level ToS flows resulting in better performance for these flows.

3.3c. Putting it all together: V-WFQ Algorithm

V-WFQ has been designed as an independent scheduling algorithm which can be used to enhance the Diffserv architecture [4]][5]. Its flow of operations is shown in Figure 16. When a packet starts at the source, all of its P-bits are set to 1. In addition, the ToS bits are determined by the ToS level corresponding to the flow. Once the packet reaches a router, the shifter module uses the PLT table to determine the appropriate value of the P-bits based on the current congestion level within the network and right-shifts the P-bits accordingly.

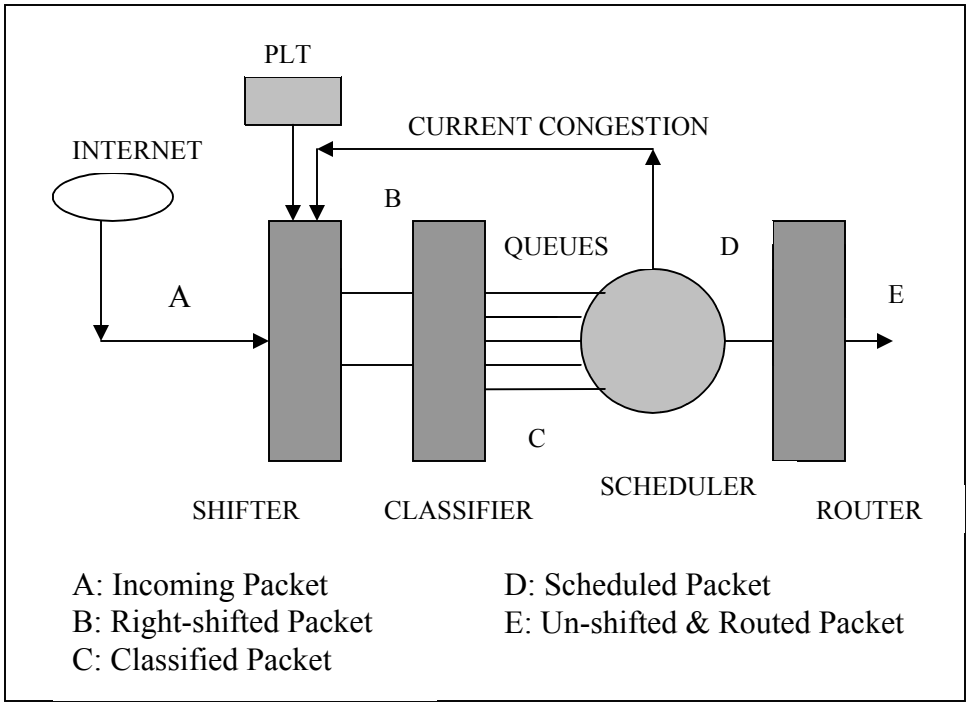

Figure 16: V-WFQ Architecture and Working

Once the P-bits are adjusted by the shifter, the packet is forwarded to the classifier, which assigns it to a specific queue depending on the new P-bits value. Next, a Weighted Fair Queuing (WFQ) [12][13] scheduler schedules the packets from each queue according to its rate, which is proportional to the weight assigned to that queue. After a router processes a packet, it is sent to the desired port and its P-bits are again set to 1.

V-WFQ requires the knowledge of the current congestion (specified as a percentage) at the router. A congestion level is computed as a function of the backlogged, or non-empty, queues of the scheduler. The congestion level at a router is calculated as the percentage of the total number of the packets in all the queues in the router to the total backlogged queue capacity (BQC) of the router. The total BQC is the summation of the queue capacity of a subset of the backlogged queues. A backlogged queue is included in the queue capacity only if it is holding significant enough packets beyond a threshold

window λ that is determined by the network administrator. For example, in our experiments we use a threshold window size of 200 packets for a 1500 packet capacity queue. The rational for this decision can be best expressed through an example. Consider a situation in which due to an increase in congestion, the low priority packets are pushed into a new backlogged queue. The backlogged queue will initially contain only a few packets. However, if this backlogged queue is included in the computation of the congestion at the router, the mostly empty backlogged queue will cause a sudden, artificial decrease in the congestion level. Therefore, a backlogged queue is not used in the computation of the congestions until it contains sufficiently large number of packets.

Using V-WFQ, as the congestion within the network increases, the gap in the percentage of resources being assigned amongst the different ToS level flows is widened, favoring the higher level ToS flows. Thus, the more congestion increases, the more resources are allocated to the higher level ToS flows. In contrast, in the WFQ algorithm, the percentage share of network resources amongst traffic flows remains constant throughout the lifetime of the network regardless of the network congestion levels.

3.3d. In order packet processing module

The V-WFQ, as explained above, could potentially process packets out of order. This Out of Order Processing (OOP) incurs when a newly arrived low priority packet gets processed through the router earlier than a previously arrived high priority packet. OOP can be better explained through an example scenario: Let's assume the congestion level is very high and a higher priority packet (P_{high}) arrives at the router. This packet will be pushed to a lower priority queue according to the V-WFQ algorithm. Now let's assume

the congestion level goes down while P_{high} is still in a low priority queue. If at this time a lower priority packet (P_{low}) arrives, it is possible that V-WFQ may allocate P_{low} to a higher priority queue than the queue in which P_{high} is stored. As a result it is possible for a newer, low priority packet to be processed sooner than an older, high priority packet.

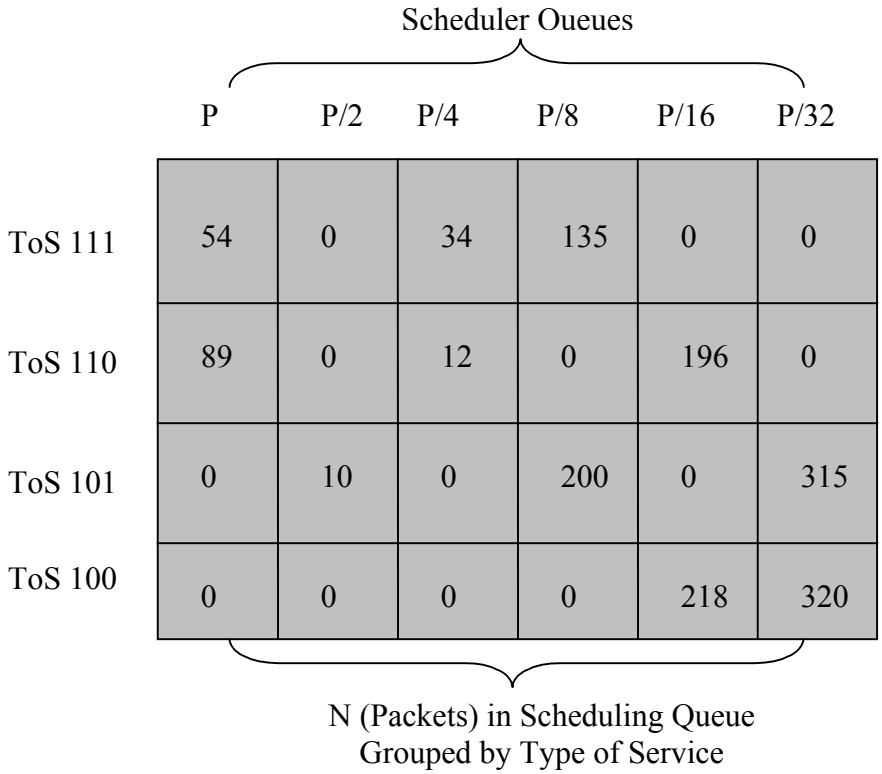

Figure 15: Preventing OOP using Shifter Look up Table

The OOP prevention is achieved by a look-up table at the router's shifter. This look-up table, an example of which is presented in Figure 17, contains as many rows as ToS levels and as many columns as the scheduler's waiting queues (in our example 4 rows for 4 ToS levels and 6 columns for queues P, P/2, P/4, …, P/32. Each entry in the shifter look-up table represents the number of packets of a certain ToS in a certain queue (implemented by a simple counter). For example, in Figure 15, there are 135 packets of ToS 111 in the P/8 queue and there are no packets of ToS 101 in the P/4 queue.

To address the OOP problem, V-WFQ uses the shifter look-up table and an additional algorithm to prevent out of order processing of packets. The idea of this algorithm, depicted in Figure 18, is to keep track of the presence of packets for each Type of Service (ToS) in the router's waiting queues. Thus, when a packet of ToS level *T* arrives, it will be assigned to a queue *q* such that there are no queues with higher priority than *q* that contain a packet of higher ToS than *T*.

```
Incoming Packet:
ToS = T, PLT entry = P/X
Algorithm:
For all queue S (A, B, C, D)
{
   If ToS (S) > T
   {
      Foreach (scheduler queue I < X)
       {If (S (I) > 0)
        {
           Send packet to P/I queue
           Where I is least possible in
         this loop
        }
       Else
        {Send packet to queue P/X as
           Determined by the shifter
        }}}
```

Figure 16: Routine to prevent Out of Order Processing

This process guarantees that any newer, low priority packet will be pushed to a lower priority queue than any queue that may contain an older, higher priority packet.

3.4. Stochastic modeling

We have designed and implemented V-WFQ as an extension of WFQ. However, in order to verify that our simulation results are accurate and consistent we will

theoretically analyze our system using Markov Chains. Furthermore, to the best of our knowledge, there has not been an in depth analytical analysis of WFQ or any of its extensions. Thus, we will provide both an analytical evaluation of WFQ as well as our extension to V-WFQ.

3.4a. Introduction to Markov Chains

Theories surrounding Markov chains are used by the researchers to analyze the performance of computer systems and computer networks [19][20]. Markov chains are a special case of Markov processes which themselves are a special case of a stochastic or random processes. Stochastic processes can be categorized in two ways, first, based on whether the values assigned to the stochastic processes are discrete or continuous over time; and second, if the indexing parameters are themselves continuous or discrete as shown in Figure 19.

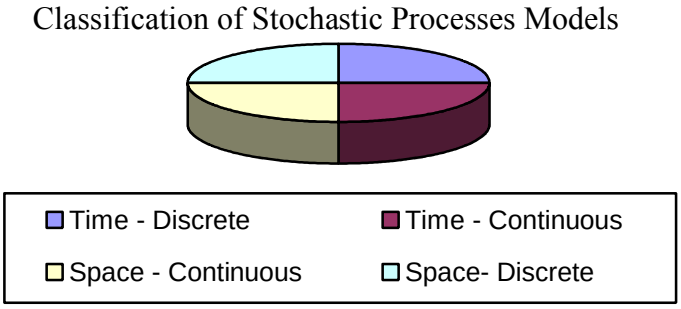

Figure 17: Classification of Stochastic Processes

A continuous set means that it can take any infinite or finite number of values in the given interval, where as discrete sets consists of finite number of possible states.

Stochastic Processes are defined as ordered sets of related random variables indexed with time *t.*

3.4b. WFQ and V-WFQ Markov chain Models

In order to analytically model V-WFQ, we employed Markov Chains to represent the workings of our system. More specifically, we replicated a two-congestion level two-priority V-WFQ system using a Markov Chain. Before we delve into the details of our implementation, let us first define both a Markov chain and the ability of the chain to be extended through its tuple representation.

A Markov chain is a discrete state random process, where the next state is only influenced by the current state. It consists of a set of states S = $\{s_1, s_2, s_3, s_4 \ldots s_n\}$. The process starts from some state s_i and continues traversing through these states. Every transition from some state s_i to s_j is associated with a probability say, $p_{ij.}$ The probability p_{ij} depends only on state s_i and not on any state which the process earlier traversed. According to [19][20], a Markov chain represents an integer time process $\{X_n, n \geq 0\}$ for which each random variable X_n, $n \geq 1$, depends on the past random variables X_{n-1}, X_{n-2}, … only through the most recent variable X_{n-1}. A transition matrix *P* is defined to store all the transition probabilities in the system. The transition matrix is always a square matrix whose size is $\{C(S) \text{ X } C(S)\}$, where *C(S*) denotes the cardinality of set S defined for the system [19][20]. A Markov chain can be logically extended to account for multiple dimensions through the representation of each state as an n-tuple. A Markov chain which is represented by a 2-tuple consists of two integer time processes $\{X_n, Y_m, n \geq 0 \text{ and } m \geq 0\}$ for which each of the random variables X_n and Y_m, $n \geq 0$ and $m \geq 0$, depend on the past random variables X_{n-1}, X_{n-2}, … and Y_{m-1}, Y_{m-2}, … only through the most recent variable X_{n-1} and Y_{m-1} respectively.

Our representation of V-WFQ for a two-congestion level two-priority system was replicated using a Markov chain as explained. First, the use of 2-tuple was necessary in order to capture the two-priority levels, a high-priority data and a low-priority data, being sent throughout the network. Let X_n represent the high priority packets in the system and Y_m represent the low priority packets in the system, where both $(n, m) \geq 0$ and $n = m$. We will use a stochastic matrix to represent the workings of our system. By definition [18], a stochastic matrix is a square matrix of non-negative terms in which the elements in each row sum to 1. Our stochastic matrix, P, will be an n x n matrix, representing the transition probabilities to each state within the network. For example, let {(0, 0), (0, 1), (0, 2), … ,(0, n), … , (1, 0), (1, 1), (1, 2), … ,(1, n), … , (n, 0), (n, 1), (n, 2), (n, n)} represent the states of the system corresponding to each of the rows and columns of P, respectively. Let the rows of the matrix correspond to the high priority packets, $X_{n,}$ and the columns of the matrix correspond to the low priority packets, Y_n. Thus, each entry in the matrix will represent an ordered pair $(X_{n,} Y_n)$, indicating the number of high and low priority packets in the system, respectively, at time n. It follows that each of the rows of our matrix, since it is stochastic, must sum to 1. Intuitively this is obvious from our representation, since each row represents the probabilities of moving to each of the states within the network, given the current state, and these probabilities must always sum to 1.

Figure 20 shows the Markov chain for a two-priority two-congestion level network. As displayed in the figure, λ_1 and λ_2 represent the arrival rates of the high-priority and low priority packets respectively. The departure rates for both the high- and low-priority packets are depicted by μ_1 and μ_2 respectively, for the 1st congestion level, while μ_1' and μ_2' represent the departure rates of the high-priority and low priority

packets respectively, for the 2nd congestion level. It should be noted that the arrival rates, λ_1 and λ_2, are independent of the congestion level. Here δ represents the time step.

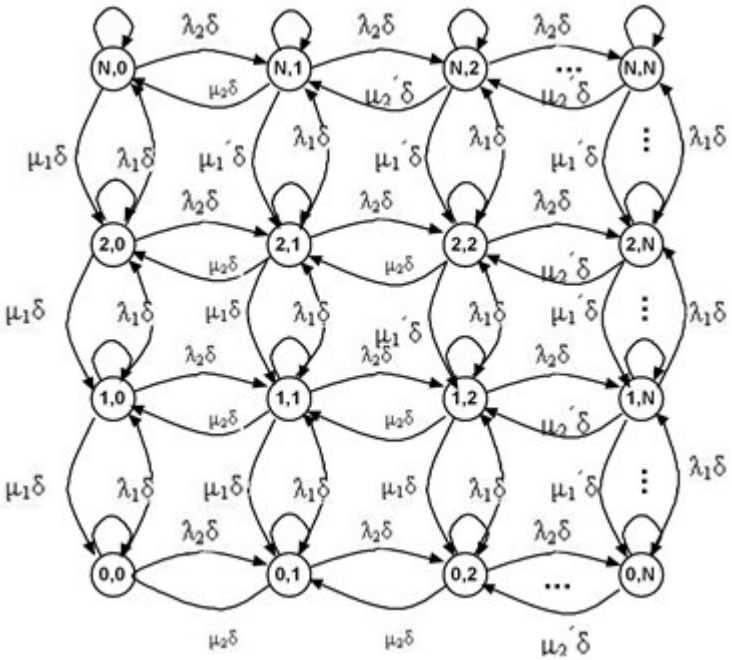

Figure 20: Two Dimensional Markov Chain for V-WFQ

$$P = \begin{bmatrix}
P_{(o,o)(0,0)} & \cdots & P_{(o,o)(0,n)} & P_{(o,o)(1,0)} & \cdots & P_{(o,o)(1,n)} & P_{(o,o)(2,0)} & \cdots & P_{(o,o)(2,n)} & \cdots & P_{(o,o)(n,0)} \cdots & P_{(o,o)(n,n)} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
P_{(o,n)(0,0)} & \cdots & P_{(o,n)(0,n)} & P_{(o,n)(1,0)} & \cdots & P_{(o,n)(1,n)} & P_{(o,n)(2,0)} & \cdots & P_{(o,n)(2,n)} & \cdots & P_{(o,n)(n,0)} & P_{(o,n)(n,n)} \\
P_{(1,o)(0,0)} & \cdots & P_{(1,o)(0,n)} & P_{(1,o)(1,0)} & \cdots & P_{(1,o)(1,n)} & P_{(1,o)(2,0)} & \cdots & P_{(1,o)(2,n)} & \cdots & P_{(1,o)(n,0)} & P_{(1,o)(n,n)} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
P_{(1,n)(0,0)} & \cdots & P_{(1,n)(0,n)} & P_{(1,n)(1,0)} & \cdots & P_{(1,n)(1.n)} & P_{(1,n)(2,0)} & \cdots & P_{(1,n)(2,n)} & \cdots & P_{(1,n)(n,0)} & P_{(1,n)(n,n)} \\
P_{(2,o)(0,0)} & \cdots & P_{(2,o)(0,n)} & P_{(2,o)(1,0)} & \cdots & P_{(2,o)(1,n)} & P_{(2,o)(2,0)} & \cdots & P_{(2,o)(2,n)} & \cdots & P_{(2,o)(n,0)} & P_{(2,o)(n,n)} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
P_{(2,n)(0,0)} & \cdots & P_{(2,n)(0,n)} & P_{(2,n)(1,0)} & \cdots & P_{(2,n)(1,n)} & P_{(2,n)(2,0)} & \cdots & P_{(2,n)(2,n)} & \cdots & P_{(2,n)(n,0)} & P_{(2,n)(n,n)} \\
P_{(n,o)(0,0)} & \cdots & P_{(n,o)(0,n)} & P_{(n,o)(1,0)} & \cdots & P_{(n,o)(1,n)} & P_{(n,o)(2,0)} & \cdots & P_{(n,o)(2,n)} & \cdots & P_{(n,o)(n,0)} & P_{(n,o)(n,n)} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
P_{(n,n)(0,0)} & \cdots & P_{(n,n)(0,n)} & P_{(n,n)(1,0)} & \cdots & P_{(n,n)(1,n)} & P_{(n,n)(2,0)} & \cdots & P_{(n,n)(2,n)} & \cdots & P_{(n,n)(n,0)} & P_{(n,n)(n,n)}
\end{bmatrix}$$

Figure 21: Corresponding Transition Matrix for Markov Chain

The matrix P, corresponding to the Markov chain in Figure 20 is shown in Figure 21. Each entry in the matrix corresponds to the probability of moving from one state to

another. For example, row 1 column 2, $P_{(0,0)(0,1)}$ represents the probability of going from state (0, 0) to state (0, 1). It can be seen from Figure 21 that this probability is λ_2. Similarly, $P_{(1,2)(1,1)}=\mu_2$.

3.4c. MATLAB and NS-2 Results

In order to illustrate both the general matrix representation, as well as a complete matrix with all of the associated probabilities, we use an example that has two priority levels (high-priority and low-priority), two congestion levels and two queues each of length 2. We will let the congestion levels be defined as either congested or not congested, where congestion is measured in terms of the number of packets currently in the system at each instance. In our sample network illustrated here, let the queue capacity of *queue*1 be *queue*1*_length* and for *queue*2 be *queue*2*_length,* both equal to 2 packets. We will designate a threshold of ≥ 3 packets in the network to indicate congestion within the network. The total number of states will be 9, since

$$number\_states = (queue1\_length+1)*(queue2\_length+1) \tag{9}$$

$$P=\begin{bmatrix}
P_{(0,0)(0,0)} & P_{(0,0)(0,1)} & P_{(0,0)(0,2)} & P_{(0,0)(1,0)} & P_{(0,0)(1,1)} & P_{(0,0)(1,2)} & P_{(0,0)(2,0)} & P_{(0,0)(2,1)} & P_{(0,0)(2,2)} \\
P_{(0,1)(0,0)} & P_{(0,1)(0,1)} & P_{(0,1)(0,2)} & P_{(0,1)(1,0)} & P_{(0,1)(1,1)} & P_{(0,1)(1,2)} & P_{(0,1)(2,0)} & P_{(0,1)(2,1)} & P_{(0,1)(2,2)} \\
P_{(0,2)(0,0)} & P_{(0,2)(0,1)} & P_{(0,2)(0,2)} & P_{(0,2)(1,0)} & P_{(0,2)(1,1)} & P_{(0,2)(1,2)} & P_{(0,2)(2,0)} & P_{(0,2)(2,1)} & P_{(0,2)(2,2)} \\
P_{(1,0)(0,0)} & P_{(1,0)(0,1)} & P_{(1,0)(0,2)} & P_{(1,0)(1,0)} & P_{(1,0)(1,1)} & P_{(1,0)(1,2)} & P_{(1,0)(2,0)} & P_{(1,0)(2,1)} & P_{(1,0)(2,2)} \\
P_{(1,1)(0,0)} & P_{(1,1)(0,1)} & P_{(1,1)(0,2)} & P_{(1,1)(1,0)} & P_{(1,1)(1,1)} & P_{(1,1)(1,2)} & P_{(1,1)(2,0)} & P_{(1,1)(2,1)} & P_{(1,1)(2,2)} \\
P_{(1,2)(0,0)} & P_{(1,2)(0,1)} & P_{(1,2)(0,2)} & P_{(1,2)(1,0)} & P_{(1,2)(1,1)} & P_{(1,2)(1,2)} & P_{(1,2)(2,0)} & P_{(1,2)(2,1)} & P_{(1,2)(2,2)} \\
P_{(2,0)(0,0)} & P_{(2,0)(0,1)} & P_{(2,0)(0,2)} & P_{(2,0)(1,0)} & P_{(2,0)(1,1)} & P_{(2,0)(1,2)} & P_{(2,0)(2,0)} & P_{(2,0)(2,1)} & P_{(2,0)(2,2)} \\
P_{(2,1)(0,0)} & P_{(2,1)(0,1)} & P_{(2,1)(0,2)} & P_{(2,1)(1,0)} & P_{(2,1)(1,1)} & P_{(2,1)(1,2)} & P_{(2,1)(2,0)} & P_{(2,1)(2,1)} & P_{(2,1)(2,2)} \\
P_{(2,2)(0,0)} & P_{(2,2)(0,1)} & P_{(2,2)(0,2)} & P_{(2,2)(1,0)} & P_{(2,2)(1,1)} & P_{(2,2)(1,2)} & P_{(2,2)(2,0)} & P_{(2,2)(2,1)} & P_{(2,2)(2,2)}
\end{bmatrix}$$

Figure 22: Generic Transition Matrix for (2, 2) Queue Length Scheduler

The generic P matrix associated with this situation is shown in Figure 22 whereas Figure 23 shows the particular values for the 9 x 9 matrix. In Figure 23, the probabilities

associated with each of the transition are shown, where the λ and μ values are the arrival and departure rates described above.

$$P=\begin{bmatrix} 1-(\sum Row0) & \lambda_2 & 0 & \lambda_1 & 0 & 0 & 0 & 0 & 0 \\ \mu_2 & 1-(\sum Row1) & \lambda_2 & 0 & \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \mu_2 & 1-(\sum Row2) & 0 & 0 & \lambda_1 & 0 & 0 & 0 \\ \mu_1 & 0 & 0 & 1-(\sum Row3) & \lambda_2 & 0 & \lambda_1 & 0 & 0 \\ 0 & \mu_1 & 0 & \mu_2 & 1-(Row4) & \lambda_2 & 0 & \lambda_1 & 0 \\ 0 & 0 & \mu_1' & 0 & \mu_2' & 1-(\sum Row5) & 0 & 0 & \lambda_1 \\ 0 & 0 & 0 & \mu_1' & 0 & 0 & 1-(\sum Row6) & \lambda_2 & 0 \\ 0 & 0 & 0 & 0 & \mu_1' & 0 & \mu_2' & 1-(\sum Row7) & \lambda_2 \\ 0 & 0 & 0 & 0 & 0 & \mu_1' & 0 & \mu_2' & 1-(\sum Row8) \end{bmatrix}$$

Figure 23: Transition Matrix for (2, 2) Queue Length Scheduler

In order to better understand, describe, and predict the behavior of our V-WFQ system, we need to be able to determine the performance of the network when it reaches a steady state. Therefore, we determine the steady state probability vector or steady state distribution, using the above matrix P displayed in Figure 23. For, finite state Markov chains, it has been previously proven, [18], that

$$\Pi = \Pi [P] \tag{10}$$

always has a probability vector solution, where π represents a row vector for the current state of the system. Furthermore, since our Markov chain has a single recurrent class, we know that it has a unique probability vector solution, [18]. Using, MATLAB we wrote a program to simulate the above scenario. Both the size of the queues, the arrival and departure rates for queues, the time interval, and the congestion threshold are user-defined and input at run-time, allowing the program to be used for a variety of different situations. Initially, π represents the starting condition for the system which will by default be,

$$\Pi = [1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0]$$

indicating that no packets are in the system when it begins. At each instance in time, defined by the user, π is updated, according to equation 3. For the previous example, let the time interval be 0.01 seconds, the arrival rate into queue 1 be 10 packets/ second, the

arrival rate into queue 2 be 15 packets/ second, the departure rate from queue 1 be 20 packets/ second, and the departure rate from queue 2 be 25 packets/second. After, running the simulation, we determined the steady state probability shown below.

$$\Pi = [0.2928 \quad 0.1788 \quad 0.1126 \quad 0.1425 \quad 0.0548 \quad 0.0661 \quad 0.0403 \quad 0.0256]$$

Hence, the probability of being in state (0, 0) is 0.2928 and the probability of being in state (2, 2) is 0.0256. Additionally, V-WFQ ensures that packets of a higher-priority get processed faster than packets of a lower-priority. Thus, we analyzed this by determining the probability of being in each state for each of the two queues separately. This can be done by calculating individual probabilities for both queues from the probability vector Π. To calculate the probability of being in state S for queue X, i.e. $P_X(S)$ can be calculated as $\sum \{\Pi(S, i)\}$, where i ranges from *0* to *number of states+1*.

The individual probability of queue 1, the high-priority queue, having all of its packets processed is 0.5841, the probability of having all but one of its packets processed is 0.2838, and the probability of having none of its packets processed is 0.1320. Similarly, the individual probability of queue 2 , the low-priority queue, having all of its packets processed is 0.5013, the probability of having all but one of its packets processed is 0.3056, and the probability of having none of its packets processed is 0.1930. These results confirm that the high-priority queue processed it packets prior to the processing of the low-priority queue as it has a greater probability of having all of its packets processed and a lower probability of having none of its packets processed. Similar results were obtained for a system with larger queue capacity.

These individual probabilities obtained for the queues were further used in calculating the mean queue length of every queue. Let us define $\in_x$ to be mean queue length of queue *x*. $\in_x$ is hence defined as,

$$Ex = \sum (Px(S) \times Number\ of\ \ packets\ \ in\ state\ S) \qquad (11)$$

where Px(S) is the probability of being in state S for queue x. In a simulation experiment in MATLAB, for 2 queues and 2 congestion levels, with every queue having a 10 packets capacity, Figure 24 shows the mean queue lengths. The experiments show the comparison between WFQ and V-WFQ algorithms in term of mean queue length. The x-axis indicates the relative service rates for different queues as the congestion level changes. As can be observed from the graphs, as the service rate for the high priority queue increases the mean queue length decreases. However, as the service rate for the low priority queue increases the mean queue length increases. Thus, the two queues reflect the inverse images of each other.

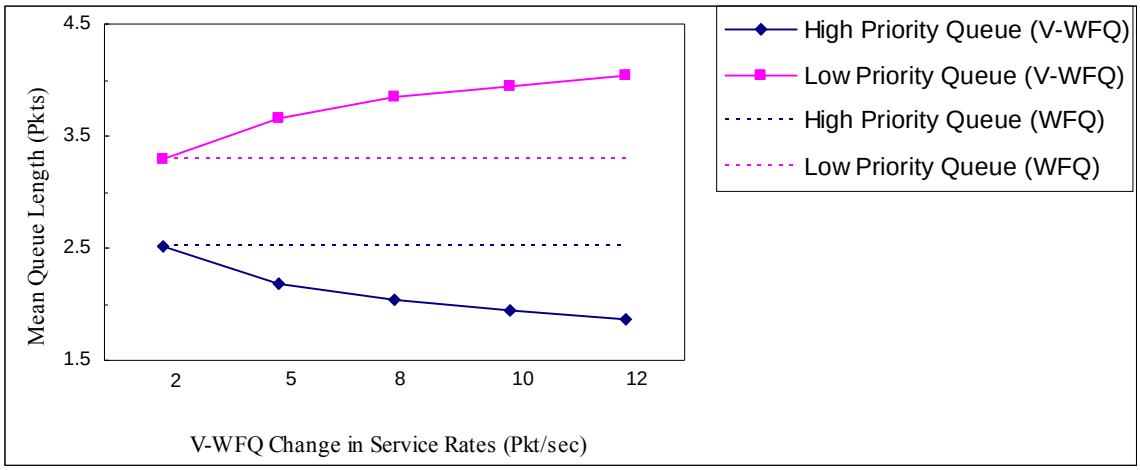

Figure 24: Mean queue length comparison for WFQ and V-WFQ

This decrease in mean queue length for higher priority queue as an increase in the relative difference in service rates occurs reflects the ability of V-WFQ to provide

better QoS as the congestion increases. The mean queue length is now used to calculate the average delay observed by packets for each of the queues using Little's law [19]. According to Little's law:

The average number of customers in a stable system (over some time interval), N, is equal to their average arrival rate, λ, multiplied by their average time in the system, T,

$$N=\lambda T$$

The behavior is entirely independent of any of the detailed probability distributions involved, and hence requires no assumptions about the schedule according to how customers arrive or are serviced. The only assumption is that the system operates in a first-come-first-served manner (FCFS) for each individual queue.

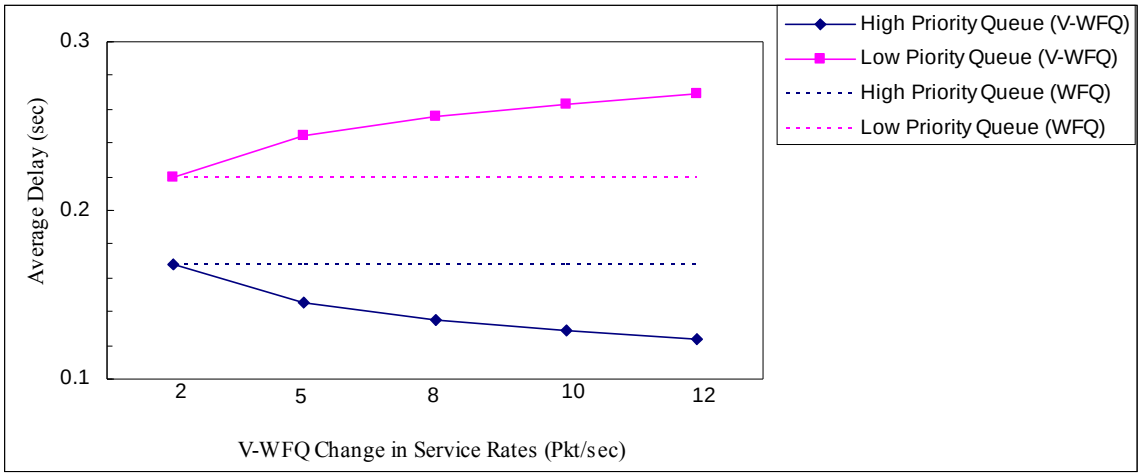

Figure 25: Average Delay comparison between WFQ and V-WFQ

Let α(t) be the arrival rate of the system in the interval [0, t]. Let β(t) be the number of departures from the same system in the interval [0, t]. Both α(t) and β(t) are integer value increasing functions by their definition. Let T_t be the mean time spent in the system (during the interval [0, t]) for all the customers during the interval [0, t]. Let N_t be the mean number of customers in the system over the duration of the interval [0, t].

If the following limits exist,

$$\lambda = \lim_{t \to \infty} \{\alpha(t)/t\} \mid \quad \delta = \lim_{t \to \infty} \{\beta(t)/t\} \mid \quad T = \lim_{t \to \infty} \{T_t\}$$

Further, if $\lambda = \delta$ then Little's theorem holds.

So the mean queue length can be converted to average delay using Little's formula as shown in Figure 25. This graph depicts the change in average delay values for each of the different levels of services as their service rate changes.

Hence after careful Markov analysis of WFQ and V-WFQ, we observe that, the performance of higher priority traffic increases significantly over the other. The two queues, two congestion levels model defined above can be further extended to a generic system. For a generic system, say consisting of *Q* queues and *C* congestion levels every state is a set with cardinality |Q|, and there are transitions based on transitions possible for different queues. There are |C| regions in the Markov chains, where the congestion level changes and hence the service rate for different queues changes. Analytical modeling of such generic system is left as a future work. However, a network system having generic number of queues and congestion level was modeled using the NS2 simulator, which explains the performance of the WFQ and V-WFQ in terms of several other parameters like the average delay, throughput, packet loss and weighted average system delay, which are explained in next chapter.

CHAPTER FOUR

EXPERIMENTS & ANALYSIS

4.1. Simulation Model

In this section we present the network structure used to evaluate the V-WFQ algorithm as compared to the WFQ algorithm based on NS-2 simulation studies. As the basis for our simulation study, we use the network structure shown in Figure 26. The servers are represent as 4 source machines: S1, S2, S3, S4 and are connected to 8 user machines, U1-U8, through routers R0, R1, and R2 that arranged in a ring topology.

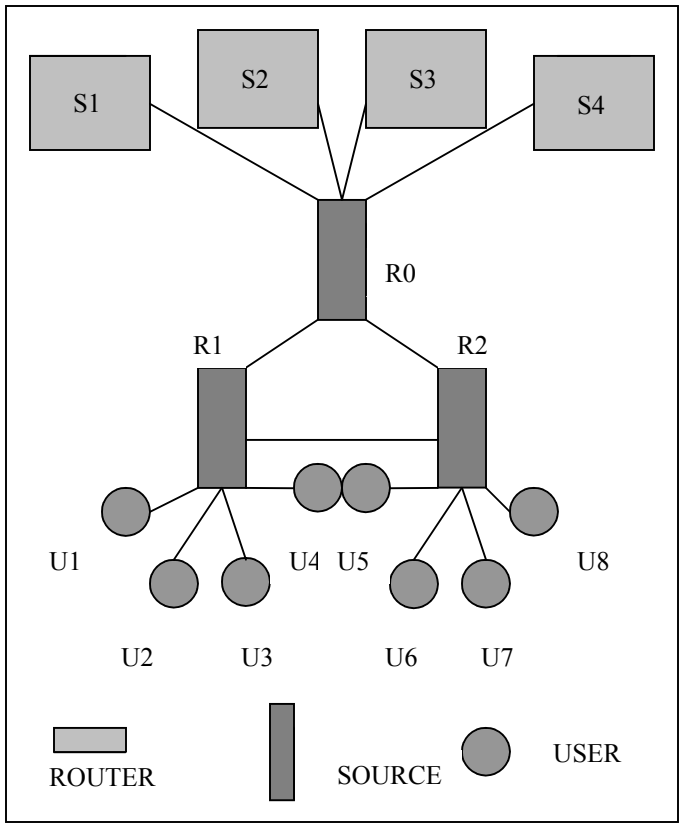

Figure 26: Network Scenario for Simulation

To compare V-WFQ and WFQ, we use the following performance metrics: average packet end-to-end delay, throughput, percentage of packet loss, and an aggregate

performance metric called Weighted Average System Delay (WASD). For a given experiment (given a system architecture, a network load, a network traffic pattern, and a time duration), the average packet delay represents the average of the all packet end-to-end delays for that experiment.

While the average packet delay is a useful metric for comparing two different systems such as WFQ and V-WFQ, it does not reflect the importance or priority of the packets. For example, the average packet delay may be the same for two different systems, but one system may provide a better quality of service for high priority packets than the other. WASD, as a performance metric, is designed to represent the system average packet delay while taking into account the relative importance among the packets according to their priority level. Let APD*i* represent the average packet end-to-end delay for all packets of ToS level *i* (or priority level *i*) and let RW*i* be the relative weight of ToS level *i* compared to all other ToS levels. For example, in Table 4, the relative weight of ToS level 111 is 0.56 while the relative weight of ToS level 101 is 0.11, indicating that ToS level 111 is 5 times more important in this system as compared to ToS level 101. For *n* ToS levels, we have:

$$\text{Average packet delay} = (\sum_{i=1}^{n} APD_i) / n$$

$$\text{WASD} = \sum_{i=1}^{n} RW_i * APD_i$$

Thus, WASD provides an aggregate measure of packet delay that takes into account the relative importance of packets based on their type of service or priority. We have performed two sets of experiments representing two types of network traffic patterns, as explained below.

Table 4: Relative Weights among ToS levels

ToS	Definition	ToS bits	Weights	Relative Weights
Level 1	Emergency Services	111	10	.56
Level 2	Administrative Services	110	5	.28
Level 3	Paid Premium Services	101	2	.11
Level 4	General Services	100	1	.05

4.2. Experiment I

Experiment I illustrates a network scenario wherein the traffic flows are evenly distributed and follow a periodic pattern. Table 5 depicts the scenario for Experiment I. A cell indicates the server-user communication with the corresponding ToS initiated during the corresponding time interval. We can see that the sources are sending out traffic in a round robin fashion; for example, in intervals T1, T2, T3, and T4 each source is sending the same ToS-level traffic to two different users.

Table 5: Experiment I scenario

Time Interval	ToS 111	ToS 110	ToS 101	ToS 100
T1: 0-30 sec	S1-U1 S1-U5	S2-U2 S2-U6	S3-U3 S3-U7	S4-U4 S4-U8
T2: 30-60 sec	S2-U2 S2-U6	S3-U3 S3-U7	S4-U4 S4-U8	S1-U1 S1-U5
T3: 60-90 sec	S3-U3 S3-U7	S4-U4 S4-U8	S1-U1 S1-U5	S2-U2 S2-U6
T4: 90-120 sec	S4-U4 S4-U8	S1-U1 S1-U5	S2-U2 S2-U6	S3-U3 S3-U7

The ToS level traffic sent out by each source to each destination is rotated in a round-robin manner, while the relationship between the source and its two destinations is

held constant. This network traffic pattern is representative of periodic, repetitive communication patterns between the users and the servers (such as automated processes for pushing financial or sports data to users).

4.3. Experiment II

Experiment II, shown in Table 6, represents a more random communication pattern in which the network traffic is unevenly generated by the servers in a non-periodic manner. Thus, the number of servers sending a particular ToS level of traffic is varied. For example, in one time interval all the servers are sending the same ToS level of traffic, whereas in another time interval the distribution is uneven.

Table 6: Experiment II scenario

Time Interval	ToS 111	ToS 110	ToS 101	ToS 100
T1: 0-20 sec	S1-U1, S1-U5 S2-U2, S2-U6 S3-U3, S3-U7 S4-U4, S4-U8	-	-	-
T2: 20-40 sec	-	S1-U1, S1-U5 S2-U2, S2-U6 S3-U3, S3-U7 S4-U4, S4-U8	-	-
T3: 40-60 sec	-	-	S1-U1, S1-U5 S2-U2, S2-U6 S3-U3, S3-U7 S4-U4, S4-U8	-
T4: 60-80 sec	-	-	-	S1-U1, S1-U5 S2-U2, S2-U6 S3-U3, S3-U7 S4-U4, S4-U8
T5: 80-110 sec	S1-U1, S1-U5 S2-U2, S3-U3 S4-U4	S3-U7	S2-U6	S4-U8
T6: 110-140 sec	S4-U8	S1-U1, S1-U5 S2-U2, S3-U3 S4-U4	S2-U6	S3-U7
T7: 140-170 sec	S4-U8	S3-U7	S1-U1, S1-U5 S2-U2, S3-U3 S4-U4	S2-U6
T8: 170-200 sec	S4-U8	S3-U7	S2-U6	S1-U1, S1-U5 S2-U2, S3-U3 S4-U4

Moreover for both the experiments, the rate of packet generation at sources are different with Rate (S1) > Rate (S2) > Rate (S3) > Rate (S4). As the ToS levels are being rotated among the sources, it is not required that the rates be in this order. This leads to every ToS level flow getting different rates at different simulation intervals which represents a more general scenario, and moreover a significant point can be observed later in the experiments which are caused due to such a relationship. Hence, every server is sending out traffic of every ToS level in different intervals in a periodic fashion.

4.4. Analyzing the experiment results

Figures 27a through 27d compare the performance of WFQ vs. V-WFQ over 10 runs of the experiment scenario I (repetitive, periodic communication). Figure 27a depicts the average packet end-to-end delay for each type of service. As noted in this figure, compared to WFQ, V-WFQ provides a shorter packet delay for high priority packets (ToS 111 and 110) while achieving longer end-to-end delays for low priority packets (ToS 101 and 100).

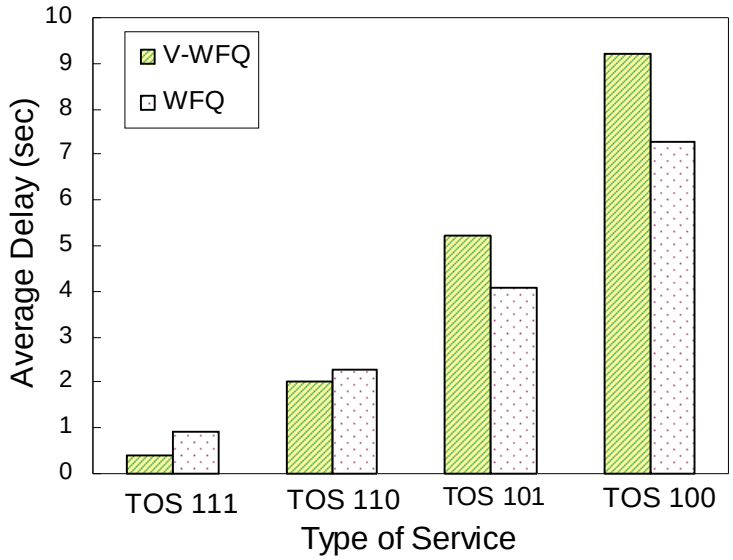

Figure 27a: Experiment I: Comparing Average End to End Delay

This result is expected as V-WFQ is designed to provide better service for high priority packets at the expense of the low priority packets. Figure 27b shows the packet drop rate between WFQ and V-WFQ in experiment scenario I. Again, as expected, V-WFQ performs better for high priority packets as compared to low priority packets (with no packet drops for the highest priority ToS level 111), but even for the low priority packets, the packet drop rates for V-WFQ are very close to those of WFQ.

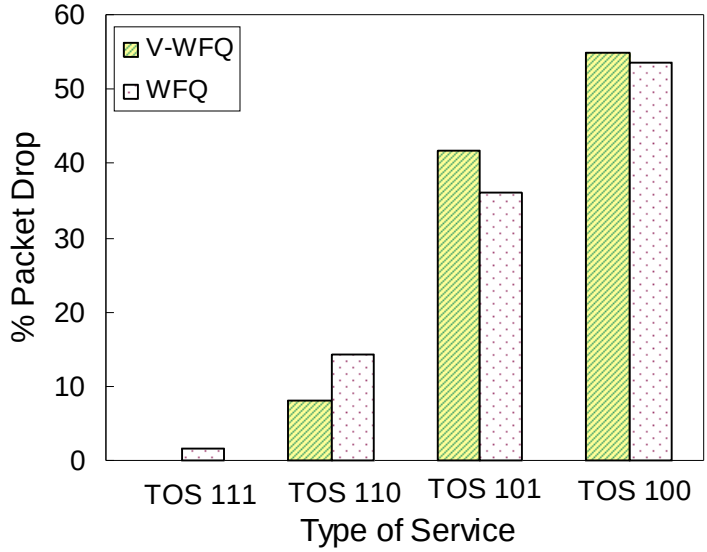

Figure 27b: Experiment I: Comparing % Packet Drop

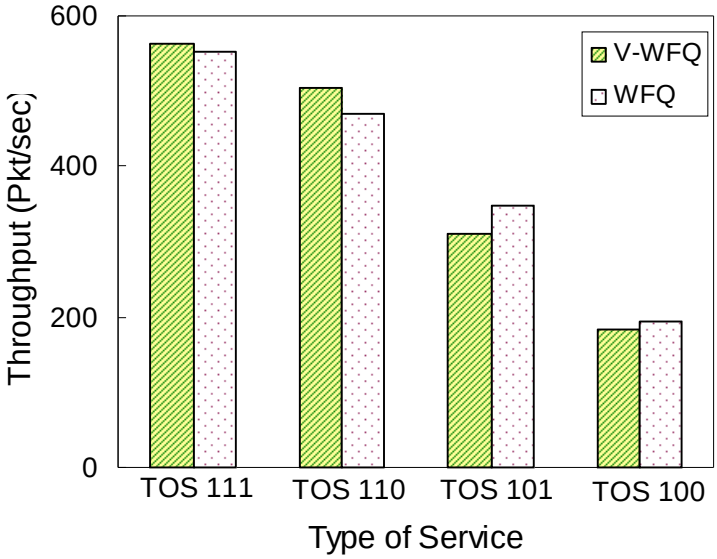

Figure 27c: Experiment I: Comparing Throughput

Figure 27c demonstrates some interesting results in terms of throughput performance. Again, V-WFQ shows superior throughput for high priority packets; but even for low priority packets the performance of V-WFQ is very close to that of WFQ.

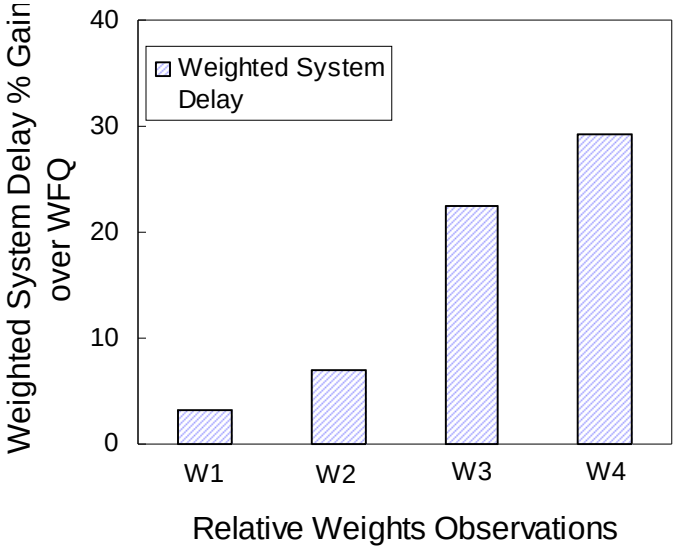

Figure 27d: Experiment I: Comparing WAS delay

The reason for good performance by V-WFQ in terms of packet drop rate and throughput, even for low priority packets has to do with the additional queues deployed by V-WFQ. These queues are used to keep more of the low priority packets active in the network while the congestion levels are high. As a result, both packet drop rate and throughput are improved for the low priority packets. Therefore, for experiment scenario I, V-WFQ outperforms WFQ for high priority packets in terms of end-to-end packet delay, packet drop rate, and throughput. For low priority packets, V-WFQ performs worse than WFQ in terms of end-to-end delay, but comparably in terms of packet drop rate and throughput. For low priority packets, V-WFQ performs worse than WFQ in terms of end-to-end delay, but comparably in terms of packet drop rate and throughput.

Figure 27d shows the percentage gain by V-WFQ over WFQ based on the WASD (Weighted Average System Delay) metric when using different relative weights for type of service levels. In Figure 27d, W1, W2, W3, and W4 represent different relative weight observations as shown in Table 7.

Table 7: Weight table used to calculate WASD

	W1	W2	W3	W4
ToS 111	0.53	0.82	0.75	0.76
ToS 110	0.27	0.10	0.15	0.09
ToS 101	0.13	0.05	0.08	0.04
ToS 100	0.07	0.03	0.02	0.01

Clearly, as we associate more importance/weight to higher priority packets, the percentage gain in average system performance for V-WFQ improves over WFQ. If we give the same weight to all packets regardless of their priority level, then the relative performance of V-WFQ compared to WFQ becomes the same as that shown in Figure 27a.

In order to better understand the system behavior, mainly in terms of the average delay, as kind and number of flows changes, we further decomposed Figures 27a into Figures 28a-28d, each of which, respectively correspond to the average delay experienced for both V-WFQ as well as WFQ over the different time intervals for a specific Type of Service. Thus, Figure 28a, displaying the average delay for ToS 111, illustrates the benefit V-WFQ has over WFQ for the first three time intervals T1, T2, and T3. For interval T4 the average delay is 0 for ToS 111. This can be interpreted from the scenario shown in Table 5, in which S4-U4 and S4-U8 traffic is being set as ToS 111 and

as the rate of arrival from sources are in the form Rate (S1) > Rate (S2) > Rate (S3) > Rate (S4).

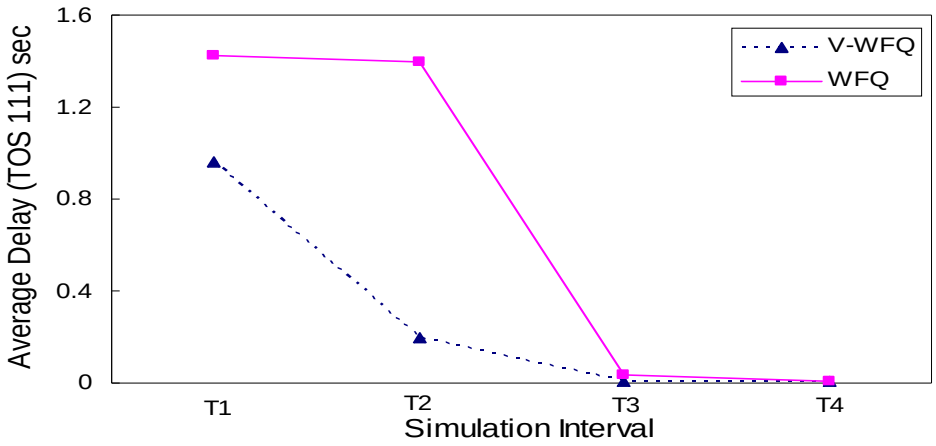

Figure 28a: Experiment I: Average Delay for ToS 111 for Discrete Simulation Interval

Therefore, S4-U4 and S4-U8 have relatively lower rates of arrival as compared to other flows. So, it causes the ToS 111 packets to get processed with nearly 0 delays in both WFQ and V-WFQ. Hence for all the discrete interval graphs shown in Figure 28a-d, we can observe that for any ToS level, the greatest delay is observed for the interval in which the flow is emerging from S1 and similarly least delay is attained when the ToS level is associated with flows emerging from S4.

For the second level of priority, ToS 110, while as expected the benefits of V-WFQ are not as substantial as they were for the highest priority traffic, V-WFQ still out performs WFQ during the entire simulation. As explained earlier, greatest and least delay are attained for interval T2, and T3 respectively as S1 and S4 gets associated with ToS 110 in these intervals which are the highest and lowest rate of arrival flows as shown in Table 5.

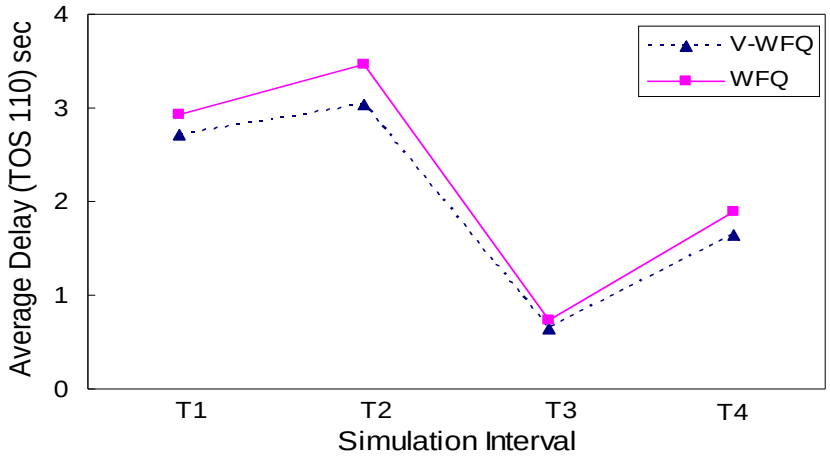

Figure 28b: Experiment I: Average Delay for ToS 110 for Discrete Simulation Interval

Figure 28c, shows the average delay as experienced by ToS 101 flow for different time intervals. As expected, the performance suffers when V-WFQ is deployed, because the system resources get biased towards higher ToS level flows. Moreover the greatest and least delays are observed for interval T3, and T4 respectively as S1 and S4 get associated with ToS 101 in these intervals which are highest and lowest rate of arrival flows as shown in Table 5.

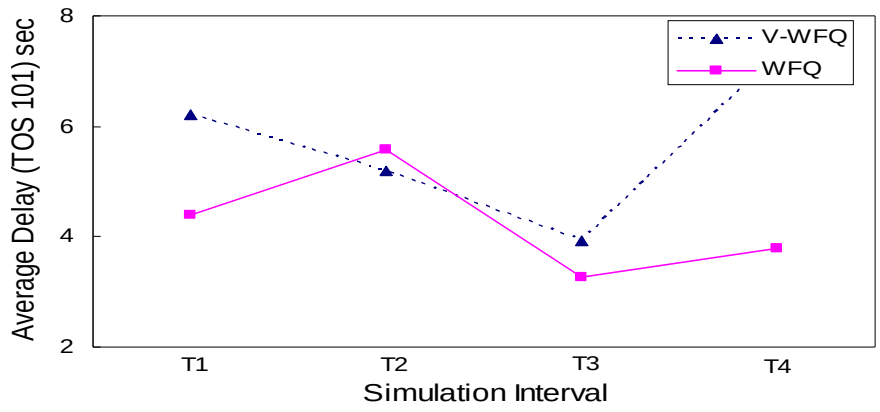

Figure 28c: Experiment I: Average Delay for ToS 101 for Discrete Simulation Interval

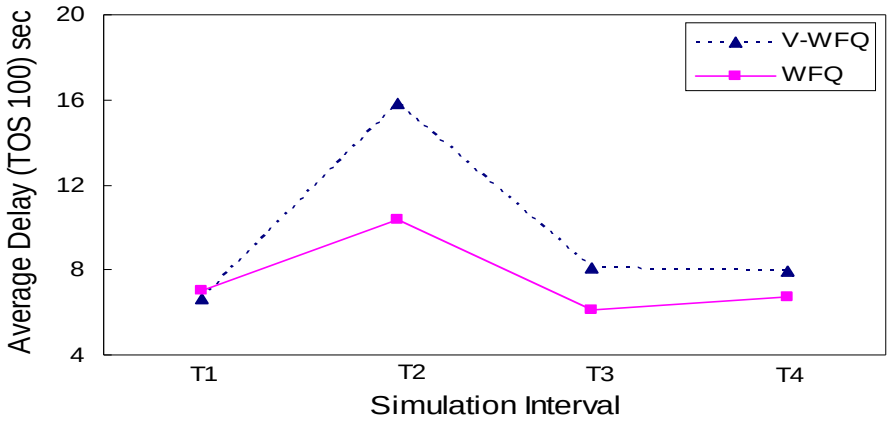

Figure 28d: Experiment I: Average Delay for ToS 100 for Discrete Simulation Interval

Similarly, Figure 28d represents the average delay for flow with ToS level 100. The performance of WFQ is quiet better than V-WFQ as ToS 100 was the lowest priority flow. The greatest and lowest delay for ToS 100 also follows the pattern explained earlier.

Figures 29a through 29d compare the performance of WFQ vs. V-WFQ, over 10 runs of the experiment scenario II (random, bursty communication). Figure 29a depicts the average packet end-to-end delay for each type of service. As noted in this figure, compared to WFQ, V-WFQ provides a better end-to-end packet delay for high priority packets (ToS 111 and 110) while performing poorer for low priority delays, especially for packets of ToS 100 (lowest level). Figure 29b shows the packet drop rate between WFQ and V-WFQ in experiment scenario II. Again, as expected, V-WFQ performs better for high priority packets as compared to low priority packets, but it is interesting to note that the packet drop rate for the lowest priority packets is almost the same for V-WFQ and WFQ.

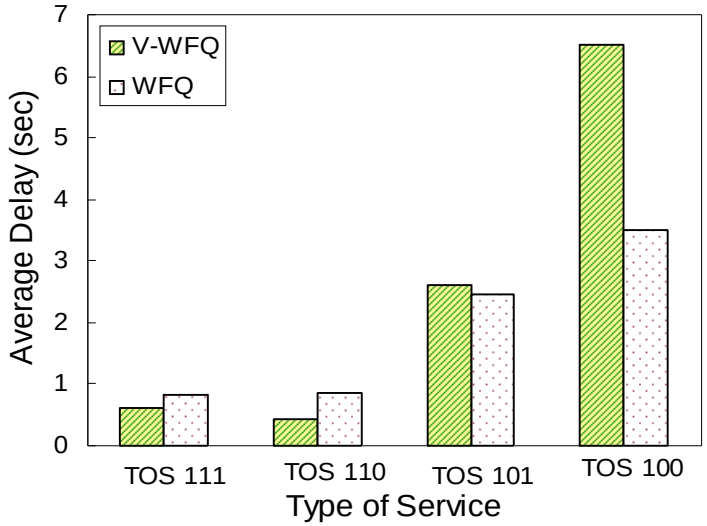

Figure 7a: Experiment II: Average Delay

Figure 29a: Experiment II: Comparing Average End to End Delay

Similarly, in Figure 29 (c) V-WFQ and WFQ perform comparably. Once again, V-WFQ outperforms WFQ for high priority packets in terms of end-to-end packet delay, packet drop rate, and throughput.

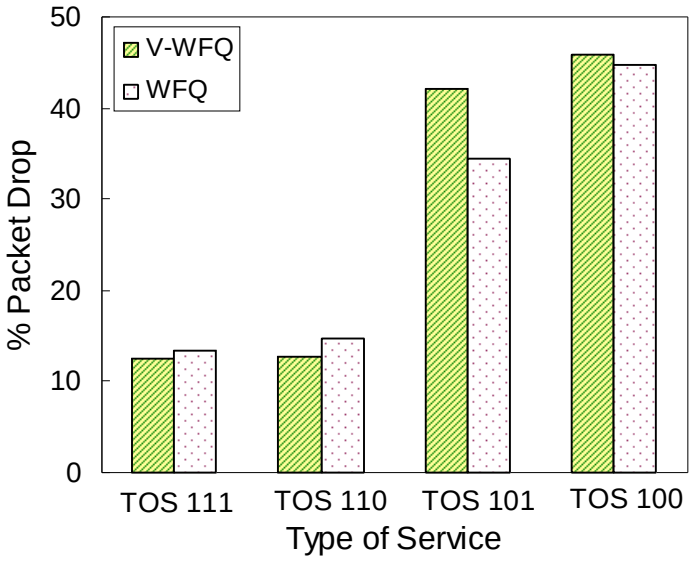

Figure 29b: Experiment II: Comparing % Packet Drop

For low priority packets, V-WFQ performs worse than WFQ in terms of end-to-end delay, but comparably in terms of packet drop rate and throughput due to the use of

additional queues to keep the low priority packets in the network when the congestion level goes up.

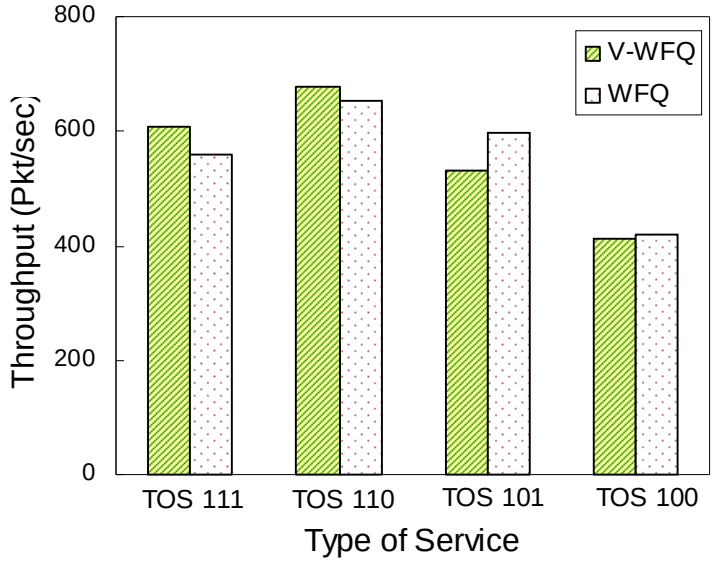

Figure 29c: Experiment II: Comparing Throughput

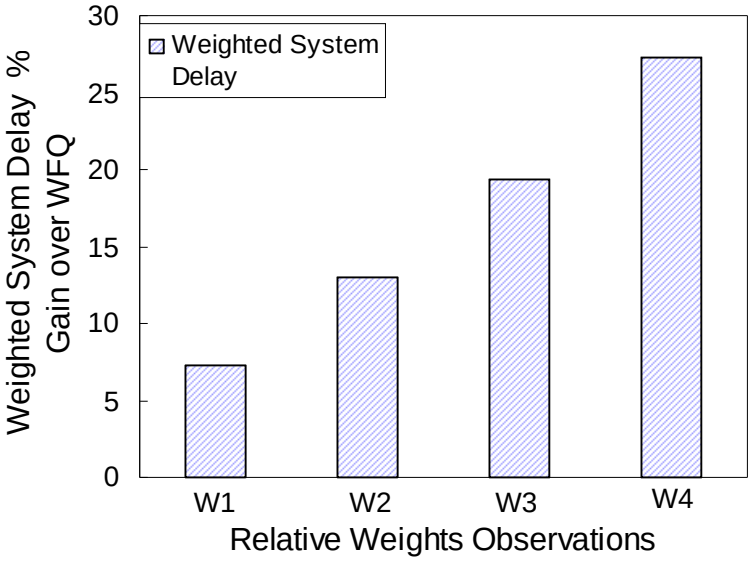

Figure 29d: Experiment II: Comparing WAS delay

Finally, Figure 29 (d) shows the percentage gain by V-WFQ over WFQ based on the WASD metric when using different relative weights for type of service levels as previously shown in Table 7.

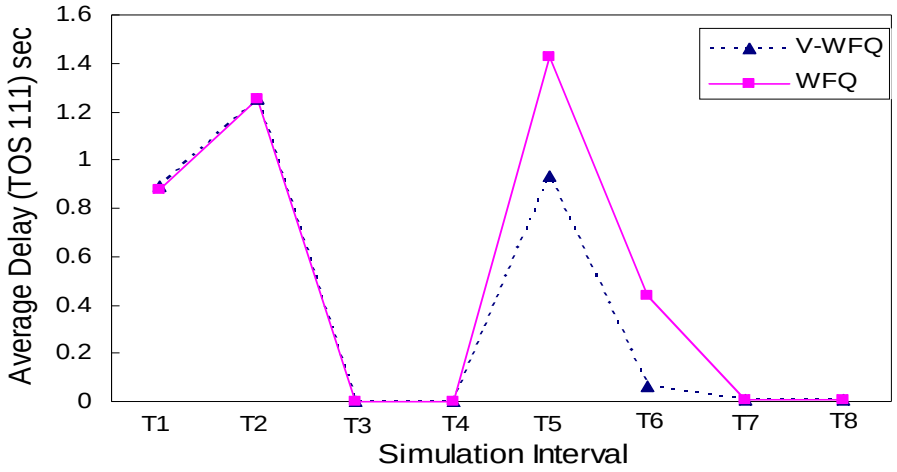

Figure 30a: Experiment II: Average Delay for ToS 111 for Discrete Simulation Interval

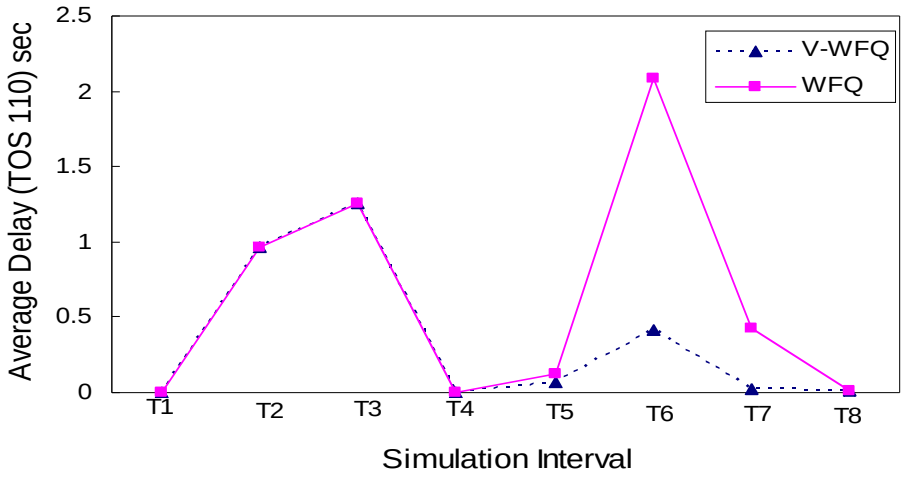

Figure 30b: Experiment II: Average Delay for ToS 110 for Discrete Simulation Interval

Figures 29a was further decomposed into Figures 30a-d, each of which, respectively, corresponds to the average delay experienced for both V-WFQ as well as

WFQ over the different time intervals for a specific Type of Service for Experiment II. The variation in the average delay for ToS 111 flow is shown in Figure 30a.

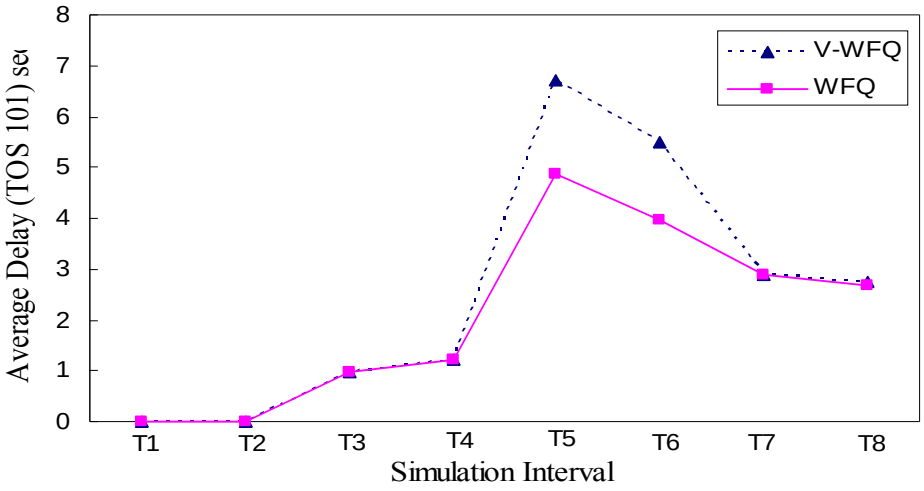

Figure 30c: Experiment II: Average Delay for ToS 101 for Discrete Simulation Interval

As shown in Figure 30a, the performance of V-WFQ is better for intervals T4- T7 and equivalent at other times. Experiment II scenario, depicted in Table 6 explains the cause of average delay being equivalent for some time intervals. As there is single ToS level flow occupying the system for some time intervals, the V-WFQ works equivalent to WFQ, as expected.

For ToS 110 also, the performance of V-WFQ is significantly better then WFQ when there are competing flows in the system as shown in Figure 30b. The performance of lower ToS level flows 101, 100 again suffers when V-WFQ is deployed in the system, particularly when there are competing flows for time interval T4 – T7. This was an expected result as the resources got biased towards the higher ToS level flows in V-WFQ.

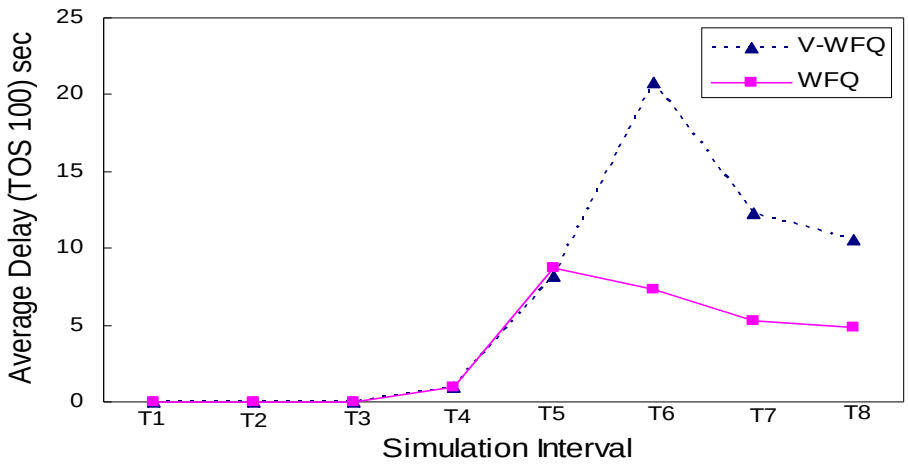

Figure 30d: Experiment II: Average Delay for ToS 100 for Discrete Simulation Interval

The performance results for both experiment scenarios I and II illustrate that V-WFQ performs exactly as intended, compared to WFQ, in terms of the end-to-end packet delay (i.e., better performance for high priority packets and poorer end-to-end delay for low priority packets). The unexpected results were that V-WFQ performed the same or better compared to WFQ in terms of throughput and packet drop rate for all packets. As explained previously, this was because of the use of additional queues in V-WFQ. It is also clearly shown that if one attaches weights (or values) to packets according to their priority level, V-WFQ achieves a better overall system value (based on the WASD metric) as compared to WFQ.

CONCLUSION & FUTURE WORK

The V-WFQ algorithm proposed in this thesis creates distinct relative priorities for different types of services at run time, based on the current congestion level at each individual router. A shifter, a Priority Look-up table, and a classification table were used in the implementation of V-WFQ. Moreover, Markov chain analysis was performed to create a mathematical model for the algorithm using MATLAB. The analytical results showed better performance for V-WFQ algorithm in terms of Average delay experienced at the router. Apart from this the algorithm was tested for several other network parameters using the NS-2 simulator. We have compared the algorithm for average end to end delay, throughput, percent packet drop and WAS delay and found V-WFQ performing better for high priority traffic.

V-WFQ requires more queues than WFQ in order to efficiently implement the switching among the queues based on the congestion level of the network. Hence, the total buffer capacity being used by V-WFQ is greater then WFQ and this resulted in comparable performance of V-WFQ and WFQ in terms of packet drop and throughput even for low priority traffics as explained earlier in the thesis. In our simulation experiments, the optimum number of queues for a 4 ToS level system was determined to be 6. Implementing V-WFQ with lesser number of queues, restricts the switching between queues, hence V-WFQ did not show significant leverage over WFQ. Meanwhile, implementations of V-WFQ with more than 6 queues induced starvation of some flows as the relative difference between the priorities of queues became too significant, and lead to no significant performance gains using V-WFQ over WFQ. Hence, if the traffic in the

network is classified into 4 ToS levels we suggest the use of a 6 queues when implementing the V-WFQ algorithm. However, the configuration can be altered by the administrator by determining the optimum number of queues for their specific network configuration. In low congestion situations the performance of V-WFQ is identical or similar to that of WFQ because V-WFQ deviates from WFQ through the use of additional queues only when congestion increases. Finally we can say that, V-WFQ enhances the performance of higher priority traffic by reallocating network resources. Hence, in a congested network where resources are limited, it provides better Quality of Service to higher priority traffic.

In this thesis work, all the experiments have been done in a simulator environment. As an extension of this work, the V-WFQ algorithm needs to be implemented in a real time sensor environment. The test bed implementation V-WFQ has already begun at Washington State University, using Sky Motes [21]. Moreover, the analytical model needs to be extended to a network or routers representing generic and complex networks.

REFERENCES

1. S. Bradner "Internet Protocol Quality of Service Problem Statement," *draft-bradner-qos-problem-00.txt,* September 1997.
2. P. Ferguson and G. Huston, Quality of Service: Delivering QoS on the Internet and in Corporate Networks, *John Wiley & Sons*, 1998.
3. X. Xiao and L. Ni, Internet QoS: A big picture*," IEEE Network*, pp. 8{18, Mar. 1999).
4. D. Verma, M. Carlson, B. Ohlman, S. Blake, Y. Bernet, J. Binder, Z. Wang, W. Weiss, E. Davies, June 1998. "A Framework for Differentiated Services," *draft-ietf-diffserv-framework-00.txt*
5. L. Fratta, F. Borgonovo, A. Capone, M. Marchese, C. Petrioli "End-to-end QoS provisioning mechanism for Differentiated Services*," draft-borgonovo-qos-ds-00.txt*, July 1998.
6. S. Blake et al, “An architecture for differentiated services,” RFC2475, Dec 1998.
7. R. Braden, D. Clark, and S. Shenker, “Integrated services in the internet architecture: an overview,” *RFC1633*, June 1994.
8. C. Dvorolis "Class-Based Service Differentiation," *draft-dovrolis-cbsd-00.txt,* June 1998.
9. J.Wroclaski, D. Clark, “An approach to service allocation in the internet" *IETF Draft*, July 1997.

10. S. Floyd, and v. Jacobson, Link-sharing and Resource Management Models for Packet Networks (compressed postscript, PDF). *IEEE/ACM Transactions on Networking*, Vol. 3 No. 4, pp. 365-386, August 1995.
11. B. Braden et al., "Recommendation on Queue Management and Congestion Avoidance in the Internet", *RFC 2309*, Apr. 1998
12. Abhay Parekh. *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*. *PhD. thesis*. Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Mass., February 1992.
13. A. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Transactions on Networking,* 2:137–150, 1994.
14. Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queuing Algorithm. *In Proceedings of SIGCOMM '89*, pages 1-12, September 1989.
15. Cisco whitepapers on Quality of Service, *http:// www .cisco.com /univercd/ cc/td/doc/ cisintwk/ito_doc/qos.htm*
16. Internet Engineering Task Force. *http://www.ietf.org/*
17. Weibin Zhao, David Olshefski and Henning Schulzrinne, Internet Quality of Service: an Overview, *Technical reports*, Columbia University.
18. A. Mrkaic., Porting a WFQ Scheduler into NS2's Diffserv Environment. *PhD thesis*. ETH, Swiss Federal Institute of Technology, Zurich, Switzerland, 2001.

19. D. Gross and C.M. Harris, Fundamentals of Queuing theory, *3rd edition, John Wiley and Sons,* 1998.

20. R.G. Gallagher, Discrete Stochastic Processes, *Kluwer Academic Publishers*, 1996.

21. Moteiv Corporation, San Francisco, CA- USA, http://www.moteiv.com/