A DYNAMIC DRR SCHEDULING ALGORITHM FOR FLOW LEVEL QOS ASSURANCES FOR ELASTIC TRAFFIC

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Sıla Kurugöl

September 2006

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Nail Akar(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof Dr. Ezhan Karaşan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. İbrahim Körpeoğlu

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

A DYNAMIC DRR SCHEDULING ALGORITHM FOR FLOW LEVEL QOS ASSURANCES FOR ELASTIC TRAFFIC

Sıla Kurugöl
M.S. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Nail Akar
September 2006

Best effort service, used to transport the Internet traffic today, does not provide any QoS assurances. Intserv, DiffServ and recently proposed Proportional DiffServ architectures have been introduced to provide QoS. In these architectures, some applications with more stringent QoS requirement such as real time traffic are prioritized, while elastic flows share the remaining bandwidth. As opposed to the well studied differential treatment of delay and/or loss sensitive traffic to satisfy QoS constraints, our aim is satisfy QoS requirements of elastic traffic at the flow level. We intend to maintain different average rate levels for different classes of elastic traffic. For differential treatment of elastic flows, a dynamic variant of Deficit Round Robin Scheduler (DRR) is used as oppose to a FIFO queue. In this scheduling algorithm, all classes are served in a round robin fashion in proportion to their weights at each round. The main difference of our scheduler from the original DRR scheduler is that, we update the weights, which are called quantums of the scheduler at each round in response to the feedback from the network, which is in terms of the rate of phantom connection sharing capacity fairly with the other flows in the same queue. According to the rate measured in

the last time interval, the controller updates the weights in proportion with the bandwidth requirements of each class to satisfy their QoS requirements, while the remaining bandwidth will be used by the best effort traffic. In order to find an optimal policy for the controller a simulation-based learning algorithm is performed using a processor sharing model of TCP, then the resultant policies are applied to a more realistic scenario to solve Dynamic DRR scheduling problem through ns-2 simulations.

Keywords: Dynamic Deficit Round Robin Scheduling, Reinforcement Learning, QoS, Elastic Traffic

ÖZET

ESNEK TRAFİK İÇİN AKIŞ SEVİYESİNDE DİNAMİK ÇİZELGELEME ALGORİTMASI

Sıla Kurugöl
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Nail Akar
Eylül 2006

En iyi çaba servisi bugün internet trafiğini taşımada kullanılmaktadır fakat bu servis hiçbir hizmet niteliği sağlamamaktadır. Hizmet niteliği sağlamak için, Tümleşik Hizmetler, Sınıflandırılmış Hizmetler ve daha yakın zamanlı Orantılı Sınıflandırılmış Hizmetler mimarileri önerilmiştir. Bu mimarilerde, gerçek zamanlı trafik gibi uyulması daha zorunlu hizmet ihtiyaci olan bazı uygulamalara öncelik tanınmıştır. Bu öncelıkli uygulamalardan geriye kalan kapasite ise esnek trafik akışları tarafından paylaşılır. Biz bu tezde, üzerinde daha önceden çok çalışma yapılmış olan gecikme ve kayıba hassas trafiğin farklı muamelesi konusu yerine, esnek trafiğin akış seviyesinde hizmet ihtiyacını karşılama konusu üzerinde durmaktayız. Bu tezdeki amacımız, esnek trafiğin değişik sınıflarının farklı ihtiyaçlarına göre istenen ortalama hız seviyelerini sağlamaktır. Bu amaç için, Önce Giren Önce Çıkar kuyruğu yerine Kalanın Sırayla Servisi (KSS) çizelgelemesi algoritmasının değişken ağırlıklı bir versiyonunu kullanmaktayız. Bu çizelgeleme algorıtmasında bütün sınıflar ağırlıklarıyla orantılı olarak sırayla hizmet görmektedir. Bizim önerdiğimiz çizelgeleme algoritmasının özgün KSS algoritmasından temel farkı, bizim algoritmamızın, her dönüşte her sıranın

ağırlığını ağdan gelen geri beslemeye göre tekrar ayarlayan bir kontrol birimi kullanmasıdır. Bu kontrol birimi önceden öğrenilmiş kurallara göre ve ağdan gelen hız bilgisi şeklindeki geri beslemeye göre her sınıfın ağırlıklarını güncellemektedir. Öncelikli üst sınıfların ağırlıkları onlara gereken kapasitelerle orantılı şekilde değiştirildikten sonra, hizmet kalitesi talep etmeyen en iyi hizmet trafiği geri kalan kapasiteyi almaktadır. Her sınıfın ağırlığını her dönüşte ağdan aldığı geri beslemeye göre güncelleyen en iyi kuralları bulmak için benzetim tabanlı bir öğrenme algoritması kullanılmıştır. Ilk olarak, bu algoritmanın Transfer Kontrol Protokolünün (TCP) basit bir modeli olan işlemci paylaşma modeli üzerinde benzetimi yapılmıştır. Bu benzetimden elde edilen sonuçlar, daha gerçekçi bir çizelgeleme senaryosunda kullanılmış ve bu senaryonun ns-2 programında benzetimi yapılmıştır.

Anahtar Kelimeler: Dinamik Kalanin Sirayla Servisi Çizelgelemesi Algoritmasi, oğrenme, hizmet kalitesi, esnek trafik

ACKNOWLEDGEMENTS

I gratefully thank my supervisor Assoc. Prof. Dr. Nail Akar for his supervision and guidance throughout the development of this thesis.

Contents

List of Figures

List of Tables

To My Family ...

Chapter 1

Introduction

Best Effort (BE) service is used to transport the Internet traffic today. In best effort delivery, the traffic is transported without commitments to users and with no additional Quality of Service (QoS) technologies implemented at edge and core nodes. With this kind of delivery, there is no guarantee of QoS. However today's applications and users require different levels of service quality to be ensured by network providers. Moreover, commercially, service providers may need to provide different QoS alternatives to users in order to increase their revenues.

QoS refers to prioritizing certain traffic types, i.e. it is necessary to prioritize vital network traffic. In QoS architectures, network resources are shared according to the need of applications that make use of network resources. Moreover, since different users require different levels of QoS, service providers need to differentiate between network traffic to satisfy different user demands and application requirements. The capacity that the customer gets can be limited by the QoS technology and users buy the necessary amount of capacity for their applications. A formal document called the Service Level Agreement (SLA) is prepared by the service providers for a service level. SLA includes service providers' commitment in terms of bandwidth, throughput, jitter, delay and methods of measurement.

The need for QoS was first discussed by IP designers when RFC791 was written in 1981 [31]. IP designers included an 8 bit field called the *type of service byte* that would be useful to provide QoS at layer 3 at that time.

On the other hand, the opponents of QoS technologies foresee that the bandwidth will be so inexpensive that the labor of managing complex QoS algorithms will be more expensive than just assigning more bandwidth to the users. In order for this view to be true, there should be no bottlenecks in the whole end-to-end network. Additionally, such kind of a network is very costly today and would be a new point of interest for hackers who could flood the network with extensive and therefore harmful traffic. Another point is that, even if great innovations have been made in optical networking technologies with Wavelength Division Multiplexing (WDM) technologies, video phones and on-demand high quality HDTV television are still not implemented today due to lack of capacity in the last mile. These examples demonstrate that there is still a need for deploying QoS at certain levels.

Internet Engineering Task Force (IETF) developed models to satisfy QoS for some applications that have more stringent QoS requirements like real time applications. Integrated Services (IntServ) model and Differentiated Services (DiffServ) model are proposed with similar objectives, i.e supporting prioritization and different levels of service. IntServ relies on QoS guarantees made on a per flow basis. Consequently, IntServ has some scalability problems even if IntServ is applied at the edge of the network where the number of flows are less than those in core nodes. Another mechanism is Diffserv, in which aggregates of flows are differentiated rather than micro-flows, hence solving the scalability problems. Despite considerable research efforts, it is still hardly used in operational environments. Also, there are some algorithms that perform relative differentiation, which means that a privileged class will have a better delay (or higher bandwidth or lower loss) compared to the best effort class. One example of relative

differentiation is Proportional Differentiated Services (PDS)[12]. In PDS, differentiation between classes is controllable and predictable, providing higher classes with better service than lower classes independent of the load conditions. One of the models of PDS depends on delay differentiation and the other on loss differentiation. In both delay and loss differentiation models, specific scheduler and dropper mechanisms are proposed in order to adjust the scheduler weights given to a traffic class such that average delay or loss ratios between classes are maintained at a desired level. The disadvantage is that there are no absolute guarantees. However, it is easier to deploy PDS since no signalling and admission control is performed as would be required in networks with absolute QoS guarantees.

The above mentioned models aim to satisfy the QoS requirements of streaming traffic by giving priority to these classes. Packets of streaming flows have priority in network queues to minimize their delay while elastic flows dynamically share the remaining bandwidth. Our aim is to satisfy some QoS assurances for elastic flows at flow level. We intend to maintain different average rate levels for different classes of elastic traffic. For the differential treatment of flows belonging to different classes, a scheduling algorithm called Deficit Round Robin (DRR) [32] is used. The reason is that, with FIFO queueing, the QoS assurance cannot be guaranteed for different classes of traffic. In FIFO queueing, when the load of the best effort traffic increases, the rates of all classes of traffic decrease simultaneously. However, using DRR scheduling, some weight, i.e. portion of bandwidth, is allocated to each class in proportion with its QoS requirement. All queues belonging to different classes are served in a round-robin fashion. Each class is served with a rate in proportion to its weight in each round. In this case, when the load of the best effort traffic increases, the rate of the other classes are not adversely affected and we obtain an average mean rate. In order to use the bandwidth resource more efficiently, the weights of the DRR algorithm are updated dynamically. By this way, when the constraints of higher class traffic

are satisfied, the rest of the bandwidth can be assigned to the best effort traffic. Therefore, the resources are more efficiently used compared to the static case.

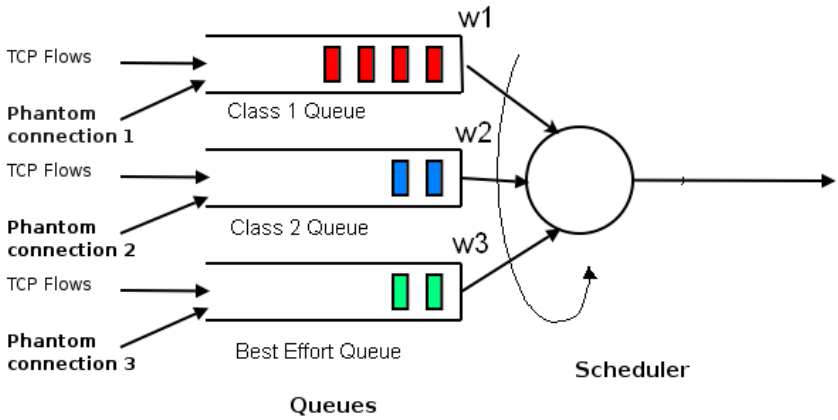

Figure 1.1: Round Robin Scheduling

In our model, we have N different service classes in addition to the best effort class. Our model describes how network resources are shared among these classes according to their QoS requirements. We aim to have a mean rate assurance r_i for the class of traffic i. In order to satisfy the QoS needs we use a dynamic scheduling algorithm at the router. In our model, we use a variant of Deficit Round Robin(DRR) scheduling algorithm but dynamically update the weights of the deficit round robin queues according to our constraints. The method is called Dynamic DRR (DDRR), hereafter. The available bandwidth resources are shared between N different classes joining the N different queues and each queue is served in a round robin fashion. However at each round, new weights are determined for each queue according to the feedback obtained from the network. By this way, optimal utilization of available resources are obtained, while assuring a rate r_i for a class i.

For the feedback mechanism, we use a feedback loop and a suboptimal policy is found to minimize the error between the current measurement and the QoS constraint r_i for each class. The QoS constraint is the desired average flow rate for each class of traffic. By using the reinforcement learning techniques

and simulating the behavior of TCP using M/G/1- PS Model, we first obtain a policy in which each class of traffic i gets an average rate of r_i. According to our control loop and using the policy obtained, the weight of the scheduler is dynamically updated at each predetermined time interval and the rate of each traffic class is observed. In order to observe the rate of each class of traffic, a phantom connection is used [2]. Flows belonging to each class of traffic join the same queue and share the resources dedicated to that queue. To measure the rate each flow gets, we add an infinite phantom connection sharing capacity fairly with other active flows of the same class. Unlike other flows, the phantom connection continuously sends dummy packets. Since phantom connections share the capacity fairly with the other currently active flows due to the way TCP operates, the rate of each flow entering a queue can be determined by observing the rate of its phantom connection. According to the rate observed for each class through phantom connections, the control loop updates the weights according to the controller parameters of the suboptimal policy obtained before. Simulations are performed and the mean rates of flows from different classes as well as the mean rates of the phantom connections belonging to these classes are observed. Different QoS constraints for different classes of traffic are satisfied while utilizing the resources more efficiently compared to a FIFO queue and static DRR cases.

The rest of the thesis is organized as follows. In Chapter 2, we present an overview of the QoS concept and different QoS algorithms proposed in the literature. Moreover, we describe different scheduling algorithms and some improved versions of these algorithms that are present in the literature. A brief overview of elastic flows and TCP mechanisms are given as well in Chapter 2. Link Provisioning using M/G/1- PS Model in conjunction with Reinforcement Learning is studied in Chapter 3. The parameters of the controller are determined by these policies. The Dynamic Deficit Round Robin algorithm is studied in Chapter 4 and the results of simulations from the previous chapter (i.e suboptimal policies)

are used to test our proposed scheduler DDRR. The final chapter is devoted to the conclusions.

Chapter 2

Background and Related Work

In this chapter we present a summary of the concept of Quality of Service (QoS), architectures for QoS and related topics such as IP traffic modelling. In the following chapters, we will present our contribution.

Services and applications over IP networks have been diversifying along with the expansion of the Internet. ISPs (Internet Service Providers) need to satisfy different QoS (Quality of Service) requirements. The users require different levels of guaranteed QoS in contrast to the best effort service, where there is no performance guarantees.

QoS technologies are developed in order to overcome the weaknesses of the best effort IP networks [4], these weaknesses can be summarized as follows:

- In case of congestion, routers response unpredictably.
- Routers can not support priority service to different service classes; all classes are treated equally.
- End-to-end service quality cannot be supported and dynamically modified.

Some algorithms are implemented at the source and at the routers to satisfy user requirements and to prevent congestion. Flow control algorithms are implemented at the source to limit the amount of traffic admitted to the network. Various algorithms are also implemented at the routers to satisfy QoS requirements using a number of queueing algorithms.

Routers play an important role for transporting packets to their destinations. Sometimes simultaneously arriving bursts may cause the resources of the router to suffer from delivering the packets immediately, so the packets are buffered at the router and delayed. The routers response in case of congestion is important to support QoS. Different packets should be treated differently in case of congestion.

2.1 QoS Architectures

2.1.1 What is QoS ?

Quality of service is a multi-aspect concept and that is why it is hard to define. According to ITU-T recommendation E.800 [19], QoS is formally defined as: "The collective effect of service performance which determines the degree of satisfaction of a user of the service". According to this definition QoS is trying to implement a service model which aims to satisfy the demands of the user or which can assure a predictable service to the users from the network. However, the demand of the users can differ with respect to different applications and users, so the attempt to satisfy these demands also needs to be differentiated.

2.1.2 Motivation for QoS

While most traffic on the Internet is delivered on a "best effort" basis, i.e., without guaranteed service, many applications require service differentiation. For instance, time sensitive applications are less tolerant to delay and delay variation and critical data traffic that requires some QoS assurances like a certain average throughout. Differentiation of service requires the ability to separate IP traffic into separate classes and then treat each class differently. It is also possible to provide bandwidth *reservations* for users using applications that required a specific amount of bandwidth for a particular period of time. For instance, an agency administrator might reserve the bandwidth needed to establish a video conference on a new policy development with staff located throughout the state. Another motivation for establishing QoS in the state's network is to enable greater control over traffic congestion.

2.1.3 QoS Models for IP Networks

Integrated Services (IntServ) Model

The Integrated Services (IntServ) architecture [9] is based on allocations of resources so as to meet the user and application QoS requirements. The reservations are made on per-flow basis so that assured bandwidth and delay can be guaranteed to each application.

The IntServ model can be described in two planes of implementation: the control plane and the data plane. The control plane is responsible of setting up the reservations. The data plane sends the data packets according to the reservations made for that flow.

The Resource Reservation Protocol(RSVP) [40], [10] is a network control protocol for establishing and maintaining Internet integrated service reservations that allows Internet applications to obtain both best-effort and real-time QoS for their data flows. Hosts and routers use RSVP to deliver QoS requests to all nodes along the path of the data stream, typically resulting in a reservation of bandwidth for that particular data flow.

One of the key components of the architecture is a set of service definitions; the current set of services consists of the controlled load and guaranteed services. Guaranteed service provides strict end-to-end latency bounds for intolerant real time traffic whereas controlled load supports nominal end-to-end latency bounds for tolerant real time and elastic traffic.

Firstly, the traffic flow is characterized on the basis of its QoS requirements. Then resource reservations are handled with a specific signaling protocol, RSVP. After the reservation set up, the reservation setup information is sent to the first router on the path. Admission is performed at the router by the routing module. Routing module determines the next hop for the reservation forwarding. Admission control process is applied by each network element along the route. Each one checks whether there are enough resources to admit the flow into its shortest path route. After a flow is admitted, the network elements at the edge of the network impose policing functions (and possibly rate shaping) on the flow. The information for the reserved flow is stored into the resource reservation table. The information in the resource reservation table is used in the data plane to configure packet scheduling and the flow identification module. The flow identification module filters packets belonging to flows with reservation and passes them to appropriate queues. The packet scheduler shares the resources to the flows based on the reservation information.

The integrated services model is not deployed in practice today. One of the reasons that have impeded the wide-scale deployment of integrated services with

RSVP is the excessive cost of per-flow state and per-flow processing that are required for integrated services. The setting of state in all routers along a path is non- scalable and non-workable administratively. In addition to the scalability problem, another problem of intserv is that it assumes that reservation state can be delivered across administrative boundaries without any problems. However, it requires complex peering arrangements among network providers.

Differentiated Services (DiffServ) Model

IntServ model, which was not feasible for implementation when there are millions of flows traversing through the network simultaneously, is simplified by pushing the complex decisions like classification of flows to edges and by restricting the set of behaviors in core routers in DiffServ model [7]. DiffServ model, which is later constructed, proposes a coarser notion of QoS. In this model, packets are marked at the edge of the network according to the performance level that they requested. Then, according to their marks, the packets are treated differently at the core nodes.

Individual flows with similar QoS requirements can be aggregated into larger sets of flows called *macroflows*. All packets in a macroflow receive the same *Per Hop Behavior* (PHB) in routers. A PHB is identified by a *Differentiated Services Code Point* (DSCP) carried within the DS field (the old IPv4 ToS Byte) in every packet. DSCP (6 bytes of DS) determines the type of service class.

Flows are aggregated into macroflows at the edge routers of a DiffServ network. One of the benefits that the aggregation provides is the scalability issue, since state is only required for a few service classes. If the number of classes is small, the per queue operations of classification, scheduling, buffer management or shaping/policing becomes simpler and faster. In addition, aggregation simplifies the network management, since the operator needs to control the service

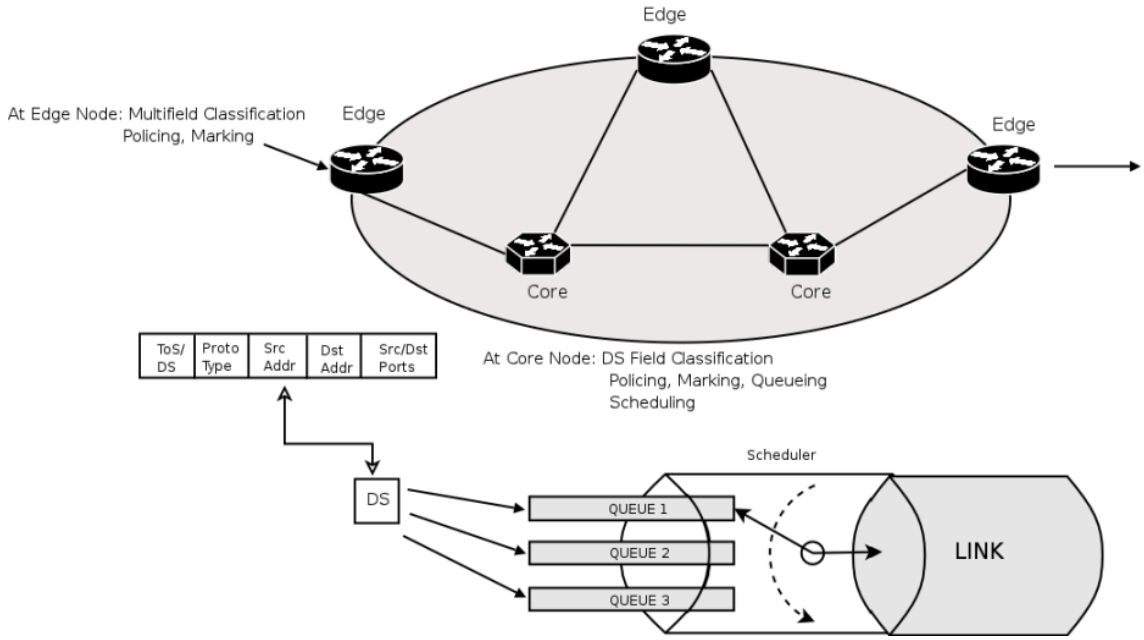

Figure 2.1: DiffServ Architecture

level of a few classes, rather than millions of flows in IntServ. However, in case of aggregation, the network is unable to guarantee a certain QoS to an individual flow.

At the edge routers, in addition to classification, marking (turning on prioritization bit values in the layer 2 and/or layer 3 headers to signify the importance of traffic), policing or rate shaping (limiting the bandwidth used on a link by queueing traffic that exceeds set rate) is also performed.

Per Hop Behavior (PHB) PHB specifies queueing, queue management such as packet drop and scheduling mechanisms. The implementer can choose different possible versions of these algorithms. For example Weighted Fair Queueing (WFQ), Weighted Round Robin (WRR) or one of the other scheduling mechanisms can be used. The PHBs in DiffServ architecture have strictly local (per hop) characteristics. So even an individual router may deploy service differentiation.

A number of PHBs were suggested for the DiffServ architecture. Best Effort is the default PHB. Other than the Best Effort two PHBs, namely, Assured Forwarding (AF) and Expedited Forwarding (EF) are standardized.

Assured Forwarding Assured Forwarding (AF) PHB is used for applications requiring better reliability than Best Effort service. AF allows more flexible and dynamic sharing of network resources, supporting the "soft" bandwidth and loss guarantees appropriate for bursty traffic. Two different classification types can be provided in the DSCP: *Service class* and *Drop precedence* of the packet. According to the service class of the packet an appropriate queue is selected for that packet and, hence, a particular bandwidth share is received from the scheduler. In AF, a packet belonging to a flow may receive three possible priority levels within the flow, which may be called drop precedences. For example sync packets must have lower loss probability since losses of sync packets result very long time-outs. Drop precedence determines the weight if the RED like queues. AF can be implemented as follows: First, classification and policing are performed at the edge routers and if the assured service traffic does not exceed the bit rate specified by the SLA, they are considered *inprofile* otherwise excess packet are considered as *out of profile*. Then, all packets, in and out, are inserted into an assured queue to avoid an out of order delivery. After that, the queue is managed by a queue management scheme like RED or RIO. Finally, queue management scheme drops or forwards the packets.

Expedited Forwarding Expedited Forwarding(EF) PHB is suggested for applications that require a hard QoS guarantee like delay and jitter. Mission critical application set a good example for this kind of service.

DiffServ Building Blocks DiffServ model includes two conceptual elements in the edge point of the network: classification and conditioning shown in figure.

Conditioning includes numerous functional elements that are used to implement conditioning actions.

Classification Packet classification identifies the packets and separates them for further processing based on the information in the packet header (See Fig. 2.3). Behavior Aggregate (BA) classifier uses only the DSCP field for classification whereas the Multi-Field (MF) classifier uses a combination of fields of the IP header (e.g. source address and source port). MF is usually used at the edges of the network for packet classification and BA in the core of the network due to its simplicity.

Conditioning Conditioning mechanisms such as metering, marking, shaping and dropping are important parts of the DiffServ Model. Conditioning is used to ensure that on average each behavior aggregate will obtain the agreed service level.

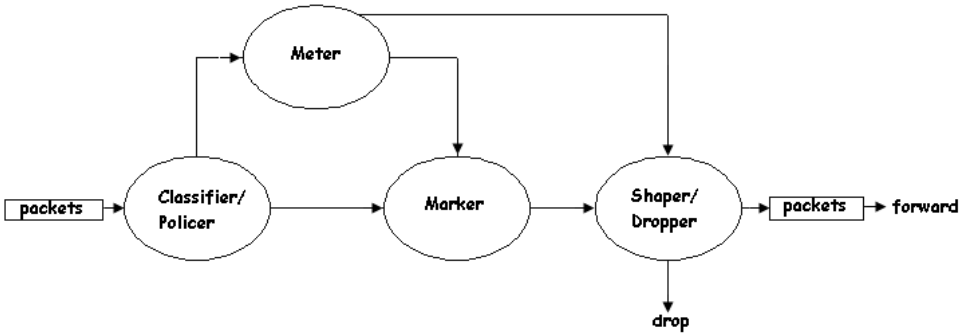

Figure 2.2: DiffServ QoS Traffic Conditioning Flow Chart

Metering is a process to determine whether the behavior of a packet stream is within the profile, i.e in profile or out of profile. There are various estimators for metering but the most known and widely used estimator in the packet networks is the "token bucket" estimator. Token bucket can be described by two parameters: token generation rate (R) and size of the token bucket (S). Each token represents

some number of bytes and the packet can be sent if enough tokens exist in the bucket. If there are not enough tokens in the bucket, the packet is either shaped or simply dropped.

Marking is a process done at the network edges where packets are marked to belong to a certain service class by setting some predefined DSCP value to the packet header to signify the importance of traffic.

Shaping limits the bandwidth used on a link by queueing the traffic that exceeds the set rate. Dropping has similar objective as shaping, but it discards out of profile packets in order to get the traffic stream to fit to a specific profile.

Active Queue Management In the routers, queues are essential as they smooth bursty traffic in order to avoid packet loss. Queue management defines the policy in which packets are dropped in case of congestion. The simplest dropping policy is drop-tail which drops incoming packets when the buffer is full. However, in case of persistent congestion drop-tail performs ineffectively and leads to higher delays, bursty packet drops and bandwidth unfairness. Hence, various active queue management (AQM) algorithms have been proposed to overcome these problems. Active queue management is a pro-active approach of informing the sender about the congestion before the buffers overflow.

Random Early Detection (RED) [13] is the most studied active queue management algorithm in the Internet, which was developed to provide better fairness, maximize the link utilization and to avoid global synchronization. RED uses the average queue size as the indication of emerging congestion. In RED, packets are dropped probabilistically as a function of the average queue size.

Scheduling Scheduling is a phenomena of deciding the order of packets to be served from different queues. Scheduling algorithms can be categorized in

number of ways. Some schedulers have the properties that make them suitable for QoS capable networks. The most important advantages of these schedulers are control over delay and jitter and rate control, when they are compared to other schedulers which serve packets whenever there is a resource available. Controlled delay and jitter is important for certain applications with hard real-time requirements (e.g. Voice over IP). Rate control enforces a traffic stream to be within its profile before forwarding it.

Scheduling mechanism alone and also as part of DiffServ architecture is very important as part of traffic differentiation and to ensure the required QoS for each class of traffic, see Fig. 2.3. Details of different schedulers will be explained later in this chapter. In addition, our contribution by proposing an alternative scheduler will be explained later in the thesis.

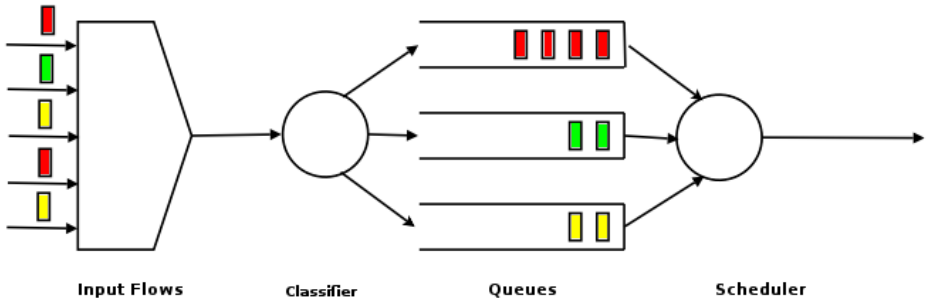

Figure 2.3: DiffServ Classification and Scheduling

Proportional Differentiated Services(PDS) Model

A more recent DiffServ model is Proportional Differentiated Services (PDS), which provides proportional services between different classes. In PDS, the differentiation between classes is controllable and predictable. Being controllable allows the network provider to adjust the QoS spacing between classes and being predictable provides higher classes with better service than lower classes independent of load conditions.

The Rate Proportional Differentiation (RPD) Model

In all of these QoS models, delay and loss requirements of streaming traffic flows are desired to be met using prioritization techniques for these flows. However our aim is to satisfy QoS requirements for elastic traffic at the flow level. One approach is [11] Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP algorithm. Weighted proportional fairness provides selective quality of service without the need for connection acceptance control, reservations or multiple queues in gateways. Moreover, as the network makes no explicit promises to the user (other than who pays more gets more) there is no need for over provisioning. The total capacity of the network is always available to its users and the price per bandwidth depends of the instantaneous demand. We have seen that the management of the receive buffers is one way to implement weighted proportional fairness when all the flows share a bottleneck and are terminated at the same host. This can be the case for example in a system of Web cache servers. Weighted proportional fairness can also be achieved by modifying TCPs congestion control algorithm. In that case the range of the weight factor seems to be limited when TCPs do not use advanced techniques like selective acknowledgement to avoid timeouts due to bursts of errors. The advantage of using the congestion control algorithm as a means to achieve weighted proportional fairness is that it can be done in a completely distributed manner and independently of where the bottlenecks are located.

2.2 Scheduling Algorithms

2.2.1 First In First Out Queueing

First in First out (FIFO) queueing is the most basic queue scheduling discipline. In FIFO queuing, all packets are treated equally by placing them into a single

queue, and then servicing them in the same order that they were placed into the queue. A bursty flow can consume the entire buffer space of a FIFO queue which causes all other flows to suffer from loss of service. Thus, FIFO queueing is not adequate; more discriminating queueing algorithms must be used in conjunction with source flow control algorithms to satisfy QoS requirements.

In order to provide QoS (Quality of Service) in high speed networks a control method at the router is needed being i.per-flow queueing, ii.Round Robin Scheduling.

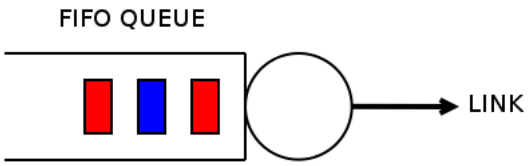

Figure 2.4: FIFO Queue

2.2.2 Fair Queueing

For the same purpose, Nagle [21] proposed a fair queueing (FQ) algorithm and Demers, Keshav and Shenker used Nagle's ideas to propose an algorithm and analyzed its performance [**?**]. In this scheduling algorithm, there exists separate queues for packets arriving from individual sources. The queues are serviced in a round robin manner. This prevents a bursty flow from arbitrarily increasing its share of bandwidth and causing other flows to suffer from congestion. When a source sends packets in a moment, it increases the length of its own queue and more packets will be dropped from that queue. The reason is that, in per-flow queueing packets belonging to different flows are isolated from each other and flows do not have impact on each other. Theoretically, one bit is sent from each flow at each round. Since this is impractical, it is suggested to calculate the time when a packet would have left the router using the FQ algorithm. After that the

packets are sorted by departure times and inserted into a queue. This algorithm can accurately guarantee fair queueing but it causes high processor loads, since it is computationally too expensive.

2.2.3 Stochastic Fair Queueing

Stochastic Fair Queueing (SFQ) is proposed in order to reduce the computational cost of FQ [28]. In SFQ, hashing is used to map the packets coming from different sources to a fixed number of queues that is fewer than the number of source-destination pairs. SFQ doesn't have a separate queue for each source destination pair with the assumption that the number of active flows at the router are much less than the total number of possible flows. Some flows are hashed into the same queue causing two connections to collide, which results in unfair share of bandwidth. If the same hash function is used, the colliding flow will collide again and get less bandwidth than they should get. In order to prevent this, the hash function is updated. Moreover, if the number of queues are larger than the number of active flows, each flow will most likely be mapped to a different queue. Therefore a relatively large number of queues will be required in order to achieve fairness.

2.2.4 Weighted Round Robin Scheduling

In weighted round robin(WRR) scheduling algorithm, there are multiple queues, each of which is serviced in a round robin fashion in proportion to its weight. In each round, each queue is visited and a number of packets, which is equal to the weight of the queue is serviced from that queue if it is nonempty. When the packet size of the flow is unknown, the weights of the wrr scheduling algorithm cannot be normalized. Therefore, the algorithm becomes unfair. Deficit round robin scheduling is used instead to overcome this problem.

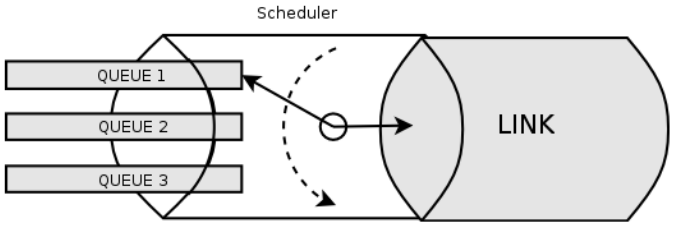

Figure 2.5: Round Robin Scheduling

2.2.5 Deficit Round Robin Scheduling

Round robin scheduling can be unfair if the flows from different queues use different packet sizes. In deficit round robin (DRR) scheduling [32], each queue gets a quantum of service in a round robin fashion. There is a deficit counter for each queue which keeps track of the portion of the quantum which is not served in that round due to the fact that the size of the forthcoming packet was larger than permitted value of bytes. The remaining bytes left from the quantum are kept at the deficit counter and in the next round, quantum is added to deficit counter and the bytes to be served are calculated.

2.3 Internet Traffic Modelling

2.3.1 Internet Traffic Differentiation

In communication networks, there are different services of traffic which belong to different applications and have different characteristics. Thus they have different performance requirements with respect to difference measures of QoS. In general some QoS measures like *transparency*, *accessibility* and *throughput* can be defined and used to distinguish different services of traffic. *Transparency* is time versus data thoroughness. For data traffic, data thoroughness is important rather than per packet delay. *Accessibility* refers to the situation of being admitted to network

or being blocked. In the internet no admission control is currently implemented. If the transfers requires a certain minimum throughput, accessibility must be considered. For data traffic the most important QoS measure is the realized *throughput*. Realized throughput depends on the provided capacity and how the capacity is shared between different flows according to the service model. In order to satisfy these QoS measures different classes of traffic, namely streaming and elastic, are defined and different service models are implemented.

Streaming Traffic

Streaming traffic is composed of flows having an intrinsic duration and rate, which is generally variable. Streaming traffic applications like audio and video require certain QoS measures to be satisfied. For example videoconferencing or voice applications are sensitive to delay.

The characteristics of streaming traffic should be known to design service models. The bit rate of long video sequences exhibits long range dependence due to the fact that the duration of scenes in the sequence has a heavy tailed distribution.

The essential traffic characteristics of the streaming flows are their *rate* and *duration*. While certain streaming applications produce constant rate flows, most audio and video applications have variable rate. Variable rate video coding produce extreme rate variations at multiple time scales. Those flows are self similar at packet level.

The number of active streaming flows depends on the time of the day. But if we take a certain busy period for example and model the arriving flows at that time as a stationary stochastic process, the traffic demand is the expected total rate of all flows in progress. This may be computed as the product of the arrival rate, mean duration and mean rate of a flow.

Streaming traffic flow durations have a heavy tailed distribution and the number of flows in progress and their combined rate are self similar processes.

The QoS of streaming traffic is frequently expressed at packet level, in terms of packet end-to-end delay and jitter. However, a flow level model can also be constructed to show some statistical bounds on end-to-end delay and jitter knowing the load induced by streaming traffic, shaping the traffic to a certain peak rate at ingress router and using preemptive priority queueing. Therefore, it is possible to provide statistical performance bounds at packet level by applying admission control at the flow level to ensure that the total load does not exceed a certain threshold [5].

Elastic Traffic

Elastic traffic is referred to as documents like data files, web page, pictures, texts,video sequences carried over TCP and stored completely before being viewed. Examples of elastic applications include e-mail, IP-based fax, applications using File Transfer Protocol (FTP) and Domain Name System (DNS). Elastic flows are mainly characterized by the size of the document to be transferred. The size of the elastic flows is extremely variable and has a so-called heavy tailed distribution, i.e. most documents are small (a few kilobytes), while the longer ones which are fewer tend to contribute more to traffic. Elastic flows care more about the average end-to-end latency. The time required to transfer a flow depends on the number of active flows on all paths that the flow passes through.

In the current Internet, elastic flows share bandwidth dynamically under the control of the TCP. The rate of a flow may vary according to the available bandwidth at that instance. Bandwidth is shared as fairly as possible among the active flows. Degree of fairness achieved by TCP depends on certain factors

like connection round trip time (RTT) and maximum window size. Bandwidth achieved by a flow depends on its size. The throughput of small flows are severely limited by the slow start algorithm of TCP. The main QoS constraint is rate, which is necessary to transfer the documents as fast as possible.

In case of normal load conditions, negligible throughput degradation can be achieved for elastic flows by fair sharing of bandwidth resources among elastic flows. In overload situations control of performance is required to avoid congestion collapse. Some techniques like admission control or other control techniques at the edge and core routers are applied to obtain a controlled performance.

2.3.2 TCP Mechanism

The TCP mechanism is described in RFC-793 [30], dating back to 1981. The transport protocol is a connection oriented and end-to-end reliable protocol designed to fit into a layered hierarchy of protocols that support Internet applications.

Connection Establishment

TCP three-way handshake mechanism is used to synchronize sender and receiver. The connection requesting instance (usually some sort of client) sends a SYN segment to the server. The server responds to the request by sending its own SYN segment and at the same time acknowledging the SYN of the client. To conclude connection setup, the client acknowledges the SYN of the server. Random initial sequence numbers are sent by both sender and receiver to synchronize sequence numbers between the endpoints. A TCP connection is uniquely identified by the 4-tuple of source and destination ports and addresses. The TCP header includes also the sequence number field for reliability. The receiver advertised window (rwnd) and the acknowledgment(ACK) fields are needed for flow control.

Flow Control

One of the most important features of TCP is flow control mechanism. Flow control prevents sender from swamping receiver with data, for example a fast server sending to a slow client. Flow control is performed by varying the size of the sliding window. Sliding window limits the amount of data that a TCP instance is allowed to send into the network without having received corresponding ACKs. The receiver advertises it receiver window (rwnd) size in ACKs. This size specifies how many more bytes the receiver is willing to accept. The sender adjusts the size of its sliding window in accordance with the size of the rwnd. The sender's sending rate is also determined by reception of ACKs sent by the receiver.

Slow Start Mechanism

The Slow Start Mechanism is a means to probe the network for available bandwidth when a new TCP connection is set up and the sender starts transmitting data. Instead of utilizing the maximum possible window size and thus injecting a larger amount of data into the network just after the connection is set up, the sender starts out slowly. First, the sender transmits only one segment. For each received ACK it increments the initial window size by one segment. This leads to an exponential increase of the window in case the receiver acknowledges every received packet.

Congestion Control

Congestion control mechanisms [3] make TCP respond to congestion in the network. The basic signal of congestion is a dropped packet which causes the host to stop or slow down.

Normally, when a host receives a packet (or set of packets), it sends an ACK (acknowledgement) to the sender. A window mechanism allows the host to send multiple packets with a single ACK as discussed under Flow-Control Mechanisms section of [3]. Failure to receive an ACK indicates that the receiving host may be overflowing or that the network is congested. In either case, the sender slows down or stops.

A strategy called additive increase/multiplicative decrease regulates the number of packets that are sent at one time. If the flow was graphed, one would see a sawtooth pattern where the number of packets increases (additive increase) until congestion occurs and then drops off when packets start to drop (multiplicative decrease). The window size is typically halved when a congestion signal occurs.

What the host is doing is finding the optimal transmission rate by constantly testing the network with a higher rate. Sometimes, the higher rate is allowed, but if the network is busy, packets start to drop and the host scales back. This scheme sees the network as a "black box" that drops packets when it is congested. Therefore, congestion controls are run by the end systems that see dropped packets as the only indication of network congestion.

At the beginning of a new connection, the TCP transmitter sets congestion window (cwnd) to one segment and sets the slow start threshold (ssthresh) equal to receiver window (rwnd). The reason is that, no other information is available about the network path at that point. The size of rwnd is an indicative of the receiver's current buffering and processing capacity. TCP grows congestion window as segments are successfully transferred to the receiver. There are two distinct growth modes; the first one is slow start (SS) and the second one is congestion avoidance (CA). In SS mode, cwnd is incremented by 1 segment for every ACK received by the transmitter. In CA mode, cwnd is incremented on the average by 1/cwnd segment. TCP connections use SS mode to ramp up

the window relatively quickly and then switch to CA mode when cwnd reaches ssthresh.

TCP has an adaptive mechanism that tries to utilize the free bandwidth on a link which is determined by the network parameters and background traffic. Full adaptation, meaning the complete utilization of the free bandwidth is not possible. The reason is that the network does not provide prompt and explicit information about the amount of free resources. TCP tests the link continuously by increasing its sending rate gradually until congestion is detected, which is signalled by a packet loss, and then TCP adjusts its internal state variables accordingly.

2.3.3 Modelling Elastic IP Traffic

TCP connections adapt their transmission rate according to the network congestion state. The TCP feedback mechanism is assumed to be ideal (i.e instantaneous feedback), then all elastic flows share link capacity equally. In [15], statistical bandwidth sharing is used to denote a form of statistical multiplexing where the rate of concurrent traffic streams is adjusted automatically to make optimal use of available bandwidth. Such sharing is achieved with a certain degree of fairness when all users implement TCP. Massoulie and Roberts [27] propose a model for a fixed number of homogeneous sources sharing a bottleneck link and alternately emitting documents. Their flow arrival process is Poisson. They identify the underlying fluid flow model as an M/G/1 Processor Sharing (PS) queue. The Poisson arrival assumption is more appropriate when the considered link receives traffic from a very large population of users. Using their approach, TCP flows sharing a link can be modelled using M/G/1 Processor Sharing. The perceived quality of service received by these TCP flows can be measured by the response time of a given document transfer, or equivalently, by the realized throughput, which is equal to the document size divided by the response time.

M/G/1 PS Model

Consider a single bottleneck link of capacity C dedicated to handle elastic flows. The elastic flows are controlled by a closed loop controller in order use the available bandwidth maximally. Assuming that the bandwidth is shared perfectly fairly by the flows in progress, a performance model is developed. In this model, it is assumed that flows arrive according to a Poisson process of rate λ and that when N flows are in progress each is served at rate C/N. Flow sizes are assumed to be independently drawn from a general distribution of mean σ bytes. With these assumptions the considered system can be recognized as an M/G/1 processor sharing queue for which a number of performance results are well known. [24] The link utilization is denoted by ρ, i.e., $\rho = \lambda\sigma/C$ and assume $\rho < 1$. Then, the number of flows in progress has a geometric distribution, $Pr[nflows] = \rho^n * (1-\rho)$ and in particular average number of flows in progress is given by $E[N] = \frac{\rho}{1-\rho}$, the expected response time of a flow of size s is $R(s) = E[\text{response time}] = \frac{s}{C(1-\rho)}$, it is proportional to the flow size. The ratio $s/R(s)$ constitutes a useful size-independent measure of flow throughput. Using Little's law, the formula for throughput can be written as $\gamma = C(1-\rho)$. When ρ is not too close to unity, the throughput is generally satisfactory for the users. The average throughput of a flow transfer can be easily expressed in analytical form and only depends on the load. Theferore, for a stable system, performance is insensitive to the flow size distribution.

Previous work on flow level QoS mechanisms for elastic traffic

In various previous work, flow level behaviour of elastic flows is analyzed and some mechanisms for QoS are proposed. The QoS mechanisms IntServ and DiffServ which were explained in the beginning of this chapter have some disadvantages. For example IntServ is not succesful in large-scale networks due to scalability and heterogeneity problems. In particular, the number of per flow

states become too large, which is the sclability issue. In addition, all nodes in the end-to-edn path must implement the same reservation protocol. An alternative protocol to overcome these problems of Intserv is DiffServ, which delivers a coarse level of QoS on a per-node, per-aggregate basis such that scalability problem is solved. However, DiffServ only provides some relative or qualitative QoS differentiation like high bandwidth, low delay or low loss by allocating more bandwidth to certain aggregates than others, or using some dropping preferences among different aggregates. The missing part of this approach is that it does not offer a quantitative QoS guarantees.

The quantitative QoS guarantees in the flow level are satisfied by static allocation methods by modelling the stochastic behaviour of flows, like flow arrivals and statistical properties of resource sharing. However, the static allocations are inefficient.

Chapter 3

Reinforcement Learning

In the next chapter the link provisioning problem will be formulated as a Markov Decision Problem (MDP) and Reinforcement Learning algorithm used to solve this problem will be discussed. Therefore, in this chapter, we provide an introduction to MDPs, their solutions via traditional dynamic programming approaches and the simulation based RL approach is given.

3.1 Markov Decision Problem(MDP)

A Markov decision process is a discrete time stochastic control process characterized by a set of states; in each state there are several actions from which the decision maker must choose. A state transition function $P_a(s)$ determines the transition probabilities to the next state by taking action a. After moving to next state, the decision maker earns a reward which depends on the new state. These states of a MDP possess the Markov property. MDP framework includes the following elements:

States: A parameter or set of parameters that are used to describe the system.

For example each location of the moving robot can be a state, or the number of people in the queue in a bank counter can be the state of the system. The transition from one state to another is random. The state space of the MDP S is composed of finite number of states $\{x_1, x_2, ..., x_N\}$.

Actions: The system moves from one state to another by performing an action. For each state s, finite number of actions are defined $A(s) = \{a_1^s, a_2^s, ..., a_M^s\}$

State Transition Probability: At each state and for each action that can be performed from that state, a transition probability of moving from that state i to the next state j by taking action a in one step is defined and denoted as $p(i, j, a)$.

Immediate Reward (or Cost): An immediate reward or cost is defined for moving from a current state to the next state under an action taken: $r(i, j, a)$ or $c(i, j, a)$.

Policy: Policy π is the rule that assigns a certain action to be taken for each state.

State Transition Time: State transition time of discrete MDPs are one. In the semi-markov decision problems, time spent in each state is another parameter $t(i, a, j)$.

Performance Metric: Each policy has an associated performance metric. Policy which has the best performance metric is is required to be found for an MDP. The performance metric can be the long run average reward (or the average cost) or the total discounted cost (or reward) calculated using a discount factor γ. The objective of the MDP is to find the policy that minimizes the average cost or discounted cost. The average cost of policy π starting at state i is defined as

follows:

$$\rho_i = \lim_{k\to\infty} \frac{E[\sum_{s=1}^{k} c(x_s, x_{s+1}) \mid x_1 = i]}{k} \tag{3.1}$$

The average cost is the sum of all immediate costs divided by number of steps taken and it is calculated on a long run. In the limit, the average cost is same for all initial states if a number of conditions are satisfied and ρ_i becomes ρ.

The policy that optimizes the value of the performance metric is the optimal policy. The optimal control is performed by the decision maker by selecting the optimal decision of that policy at each state.

Bellman optimality equation is one of the fundamental results showing the existence of an optimal policy for an MDP when certain conditions are met. The Bellman optimality equation is given by

$$V^*(i) + \rho^* = \min_a [c(i,a) + \sum_j p(i,a,j)V(j)] \tag{3.2}$$

Selecting the action that minimizes the right hand side is average cost optimal. In the equation $V^*(i)$ is the value of state i, i.e. the total minimum average cost(or maximum average reward) one can get beginning from that state and $c(i,a)$ is the expected immediate cost of taking action a at state i. ρ^* is the average one step cost (reward). The value of the current state plus the cost for one step should be equal to immediate costs plus the expected value of next state.

The Bellman Theorem is given as follows:

Theorem 1. Considering average cost for an infinite time horizon for any finite unichain MDP, there exists a value function V^* and a scalar ρ^* satisfying the system of Bellman equations for all $i \in S$, such that the greedy policy π^* resulting from V^* achieves the optimal average cost $\rho^* = \rho^{\pi^*}$, where $\rho^{\pi^*} \geq \rho^{\pi}$ over all policies π.

Greedy policy mentioned in Theorem 1 is the policy constructed by choosing the actions minimizing the right hand side of Bellman's equation.

3.2 Dynamic Programming

The systems modelled as Markov Decision problem (MDP) can be solved by Dynamic Programming (DP) methods. There are two approaches in this framework. The first one is iteratively solving the linear system of Bellman's equations, which is called the policy iteration method. The second one is using the Bellman transformation in an iterative style to compute the optimal value function, which is called value iteration. These methods require the exact computation of transition probabilities. A detailed analysis of these algorithms can be found in [6].

The algorithms to solve MDPs has the following two kinds of steps, which are repeated in some order for all the states until no further changes take place:

$$\pi(i) = \arg\min_a [c(i,a) + \sum_j p(i,a,j)V(j)] \tag{3.3}$$

$$V(i) + \rho = c(i) + \sum_j p_\pi(i,j)V(j) \tag{3.4}$$

3.2.1 Policy iteration

In policy iteration (Howard 1960), step one is performed once, and then step two is repeated until it converges. Then step one is again performed once and this goes on until convergence. Instead of repeating step two to convergence, it may be formulated and solved as a set of linear equations. This variant has the advantage that there is a definite stopping condition: when the array π does not change in the course of applying step 1 to all states, the algorithm is completed.

The algorithm can be summarized as follows:

1. Initialize number of iterations $k = 0$ and set initial policy π^0 to some arbitrary policy.

2. Given a policy π^k, solve the set of $|S|$ linear equations of above equation 3.4 for the average reward ρ^{π^k} and relative values $V^{\pi^k}(i)$. This step is called *policy evaluation*.

3. Given a value function $V^{\pi^k}(i)$, compute an improved policy π^{k+1} by selecting an action minimizing the right hand side of equation 3.3 above. This step is called *policy improvement*.

4. If π^{k+1} is different from π^k go to step 2, otherwise stop.

3.2.2 Value iteration

The policy iteration algorithm requires solution of $|S|$ linear equations at every iteration, which becomes computationally complex when $|S|$ is large. An alternative solution methodology is to iteratively solve for the relative values and average reward, which is called value iteration method.

In value iteration (Bellman 1957), the two steps of calculation of $\pi(i)$ and calculation of $V(i)$ are combined by substituting the equation of $\pi(i)$ into the calculation of $V(i)$. The algorithm can be summarized as follows:

1. Initialize $V^0(i) = 0$ for all states i, specify an $\epsilon > 0$ and set $k = 0$.

2. For each $i \in S$, compute $V^{k+1}(i)$ by

$$V^{k+1}(i) = \min_a [c(i,a) + \sum_j p(i,a,j)V^k(j)] \tag{3.5}$$

3. If $\text{sp}(V^{k+1} - V^k) > \epsilon$, increment k and go back to step 2. Here sp denotes span, which is defined as follows for a vector x: $\text{sp}(x) = \max_{i \in S} x(i) - \min_{i \in S} x(i)$.

4. For each $i \in S$, choose $\pi(i) = a$ that minimizes $[c(i,a) + \sum_j p(i,a,j)V^k(j)]$ and stop.

In value iteration algorithm, the values $V(i)$ can grow very large and cause numerical instabilities. A more stable version, called relative value iteration algorithm is used in practice. This algorithm chooses one reference state and value of that reference state is subtracted from value of all other states in each step 2.

3.3 Reinforcement Learning

Reinforcement Learning [16] is a simulation-based technique for solving MDPs where the optimal value function is approximated using simulation. Classical dynamic programming algorithms, such as value iteration and policy iteration, can be used to solve these problems if their state-space is small and the system under study is not very complex. Otherwise, these algorithms can not be used since these algorithms require the computation of the one-step transition probabilities. If the system stochastics are very complex, it is difficult to obtain expressions for these transition probabilities. If the state space is large, the number of transition probabilities is too large, therefore it is not possible to even store them.

Reinforcement learning (RL) is a simulation-based method to solve large-scale or complex MDPs since, in RL, the transition probabilities are not computed. When the state-space is large, function approximation scheme such as regression or a neural network algorithm can be used in RL to approximate the value function.

In RL, there is a learning agent, that makes the decisions and there is an environment which gives responds to the these decisions or actions of the agent.

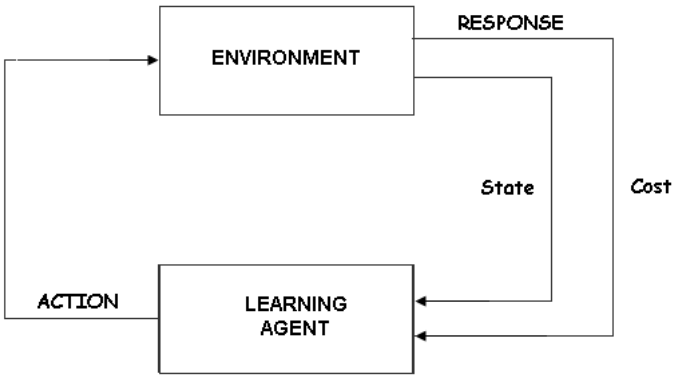

Figure 3.1: Agent Environment Interaction

RL [36] is learning what to do, how to map situations to actions so as to maximize a certain reward. By trial-and-error, the learning agent finds the actions that yields the largest reward. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Trial-and-error search and delayed reward are the two most important characteristics of RL. RL model can be summarized as follows: The agent is connected to the environment via actions and after each step of choosing an action, the agent receives a feedback from the environment about the current state of the environment and a scalar reinforcement signal that is a result of its action. The agent's aim is to choose actions so as to maximize the long run average of values of this reinforcement signal. The knowledge base is made up of values called Q-factors for each state-action pair, shown as $Q(i, a)$. These Q-factors may be in terms of cost or reward. Before learning begins all Q-factors are initialized to the same value. In each decision making step i, the agent checks $Q(i, a)$ values for all a and selects the action a^* with the minimum cost (or maximum reward). Then the response of the system to this action is simulated and the system moves up to another decision making state j. During this transition from state i to j, the system gathers information from the environment which will be given as a feedback in terms of the immediate costs incurred to the agent. The agent uses this feedback information to update

$Q(i, a^*)$. However, choosing the minimum cost action at each state may lead to a wrong policy since, a short term effect of an action may shadow or emphasize the other possibly better or worse actions. So it is necessary to try all actions of that state. For this purpose, the agent sometimes diverts from its most preferred action to another action, which is called *exploration*. After a large number of iterations, the most preferred action becomes clearer and the others divert from it. So a near optimal policy is obtained.

3.3.1 Relating RL to Dynamic Programming

In Dynamic programming, the state transition probabilities are needed to be calculated to find the optimal solution. In RL formulation, we want to eliminate these probabilities.

3.3.2 Q-Learning

Q-Learning algorithm is basically solving the Bellman equation iteratively in an asynchronous style so as to obtain the optimal value function. In the Q-Learning algorithm by Watkin [38], the state transition probabilities are eliminated. The cost function is defined as discounted reward. The average cost version of the algorithm is given as follows: All the $Q(i, a)$ values are initialized to some value. At iteration t, the learner either chooses the action a with the maximum $Q_t(i, a)$ value or selects a random *exploratory* action. This action results of transition from state i to state j and the agent receives an immediate reward $r_{imm}(i, a, j)$, and the current $Q_t(i, a)$ values are updated as follows:

$$Q_{t+1}(i, a) \leftarrow (1 - \alpha)Q_t(i, a) + \alpha(r_{imm}(i, a) + \min_a(Q_t(j, a))] \tag{3.6}$$

where $0 \leq a \leq 1$ is the learning rate controlling the contribution of new knowledge to the old one. Q-learning asymptotically converges to the optimal policy

for a finite MDP. The convergence conditions are given in [37] and can be summarized as follows: All state action pairs must be visited infinitely often, and the learning rate must slowly decay to zero.

3.3.3 Gosavi's RL Algorithm

Gosavi's RL algorithm can be applied to Semi Markov Decision Processes as well. So there is an additional variable, which is the state transition time $t(i, a, j)$. For the proof of convergence and details, refer to [16]. The algorithm can be summarized as follows from the Gosavi's tutorial [17] :

- *Step 1*: Set Q-factors to some arbitrary values, e.g to 0:

$$Q(i, a) \leftarrow 0 \text{ for all } i \text{ and } a \ . \tag{3.7}$$

  Set iteration count $k = 0$, cumulative cost $c_{cum} = 0$, total time $T = 0$. ρ^k is the average cost in the kth iteration. Set $\rho^0 = 0$ or set to a guessed value of optimal average cost. Select a first state i to start the simulation. Let α^k denote the main learning rate in the kth iteration. Set α^0 to some arbitrary value. Let β^k denote the secondary learning rate. Set β^0 to a value smaller than α^0. $ITERMAX$ is the number of iterations, and it should be set to a large number. β^k and α^k are positive decreasing functions of k.

- *Step 2*: With probability $(1 - p)$, choose an action $a \in A(i)$ that minimizes the cost $Q(i, a)$, otherwise choose a random(exploratory) action from the set $A(i)$ different from a.

- *Step 3*: Simulate the chosen action. Let the system state at the next decision epoch be j. $t(i, a, j)$ is the transition time to the next state j, and $c_{inc}(i, a, j)$ is the immediate cost incurred in the transition resulting from performing action a in state i.

- *Step 4*: Perform the update:

$$Q(i,a) \leftarrow (1-\alpha)Q(i,a) + \alpha\{c_{inc}(i,a,j) - \rho^i t(i,a,j) + min_a Q(j,a)\} \quad (3.8)$$

- *Step 5*: If an exploratory action was chosen in step 2, go to step 2, go to step 6, otherwise perform:
  - Update total cost: $C \leftarrow (1-\beta)C + \beta c_{inc}(i,a,j)$
  - Update total time: $T \leftarrow (1-\beta)T + \beta t(i,a,j)$
  - Update the average cost as: $\rho^{k+1} \leftarrow (1-\beta^k)\rho^k + \beta^k \frac{C}{T}$
- *Step 6*: Set current state i to new state j, and increment k by 1. Stop if $k = ITERMAX$, else go to Step 2.

For MDPs set all $t(i,a,j) = 1$. When the algorithm terminates, policy is evaluated from the relation:

$$\pi(i) = \arg\min_a Q(i,a) \quad (3.9)$$

The learning rates α, β should decay to 0 in order for the algorithm to converge.

3.3.4 Exploration

Exploration is very important for guaranteeing the convergence of RL algorithms [36]. Especially it is required that all of the state-action pairs (s,a) are infinitely often visited for convergence of the algorithms. Therefore, optimal (minimum cost) action is not chosen at each state, instead an exploratory action is chosen according to the exploration method. There are different exploration methods, which can be found in [36].

In our simulations we use the basic $\epsilon-$greedy exploration: Instead of selecting the action with the smallest Q value, select a random action with probability ϵ.

Chapter 4

Link Provisioning using M/G/1-PS Model in conjunction with Reinforcement Learning

4.1 Introduction

In this thesis, our main objective is to adapt the allocated resources according to the different demands of different users automatically, i.e. different QoS requirements. Therefore classification of traffic according to the requirements of the users is necessary. Thus a subscriber that is willing to pay more could benefit smaller delays and larger throughput. Our objective is to guarantee a certain average throughput for each class of traffic entering different queues at the router. We will propose an optimal scheduling algorithm to satisfy our objective. We use an adaptive version of deficit round robin scheduling algorithm for this purpose. As an introductory problem, we study a link provisioning problem and use the results obtained from this problem to determine the weights of the scheduler for our main problem. These connections will be presented in the following chapter.

This chapter is devoted to introductory Link Provisioning problem. MDP formulation of the problem will be explained. In the formulation of the problem, modelling of TCP using processor sharing approach is used and this model will be explained. The solution approach constructed based on Gosavi's RL Algorithm using simulations will be described as well. Finally, RL simulations that are performed to obtain the suboptimal policies and their results will be presented.

4.2 Link Provisioning

Link Provisioning problem is assigning certain bandwidth resources to a user according to the QoS requirements of that user. In static bandwidth provisioning, a user purchases a fixed amount of bandwidth a priori from the service provider for each pipe used to connect to the service gateways. In dynamic bandwidth provisioning, in addition to purchase bandwidth a priori, it is also possible to dynamically request for additional bandwidth to meet QoS demands, and pay for the dynamically allocated bandwidth accordingly.

Our aim is to meet some QoS demands at the flow level, while using the bandwidth resources most efficiently. The QoS constraint of the users is to obtain a certain average rate at the flow level. A certain fixed amount of resource can be allocated to the user according to the demand. However, this static allocation causes the best effort traffic to starve even when the higher priority class is not utilizing the resources. Also, fixed allocation causes the rate of higher priority class to fluctuate excessively. Therefore, dynamic bandwidth provisioning will be performed. In order to use the bandwidth resources efficiently while meeting QoS constraint of higher priority user, we need to solve an optimization problem. The capacity allocated to the link is needed to be optimized so as to satisfy the QoS constraints with minimum amount of capacity. The remaining capacity can be used by the others using that link.

The dynamic link provisioning problem needs to be formulated and solved. The system is modelled as in Fig. 4.1. The bottleneck link (access line) connects the customer lines to the core network via the router. Since the majority of internet traffic is TCP, we assume that the link is carrying only TCP traffic. Assuming that the TCP feedback mechanism is ideal, a network link carrying only TCP traffic can be modelled using M/G/1 Processor Sharing (PS) Model as described at the end of Chapter 2. Therefore, assuming elastic traffic is entering the router at the bottleneck link, our system can be modelled by an M/G/1-PS model.

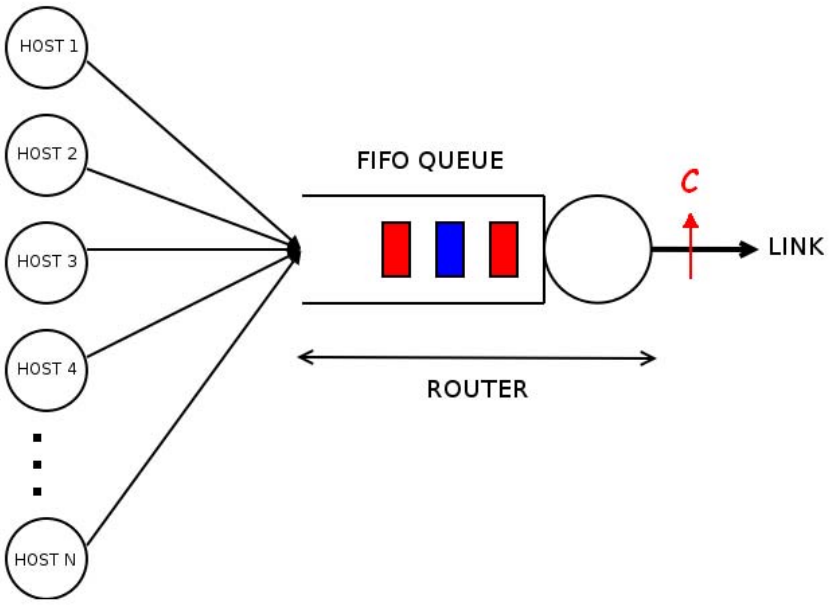

Figure 4.1: System Model

4.3 Scenario

The scenario for dynamic bandwidth provisioning problem is explained as follows: We consider a single bottleneck link with certain number of TCP connections transported over this link. Our objective is to guarantee a certain average throughput for each connection while minimizing the dynamic capacity allocated

to bottleneck link. The bottleneck link capacity is updated using a closed loop control system. According to the feedback obtained from the network, the capacity of the link is updated. This feedback is of the form of the rate of an additional connection, called the infinite phantom connection, sharing capacity fairly with other connections according to the M/G/1-PS model [23]. This feedback from the network is the measure of flow rate at that moment. So the phantom connection proposed by Afek [2] is used as a method to measure the flow rate of TCP flows sharing the capacity equally according to M/G/1-PS model. Phantom connection sends a continuous stream of dummy packets and reacts to packet loss exactly as a regular TCP connection. Since all TCP flows share capacity fairly according to M/G/1-PS model, the rate that each flow gets can be obtained by calculating the rate of phantom connection. Its rate is measured by counting the number of packets arriving from phantom connection in fixed time intervals of length δ seconds and multiplying the number of packets by packet size and dividing this number by δ.

A suitable but simple control algorithm, Proportional (P) controller [35] can be used for dynamic capacity adaptation. P controller works in a closed-loop system shown in Fig. 4.2. The variable (e) represents the tracking error, the difference between the desired input value T_{set} and the actual output (T). This error signal (e) will be sent to the P controller, and the controller computes the input to the system. The relationship between the variable (e) and (u) is that $u = K_p e + D$, i.e., proportional controller plus a fixed control. With regards to the dynamic bandwidth provisioning problem, the variable (u) is the capacity allocated to the aggregate of flows, (T) is the measured smoothed throughput of the phantom connection, and (T_{set}) is the desired throughput for individual TCP connections. Performance of the closed loop control system using a proportional controller is studied by simulations using the M/G/1-PS model. However, first fine tuning of the controller parameters are necessary in order to achieve improved

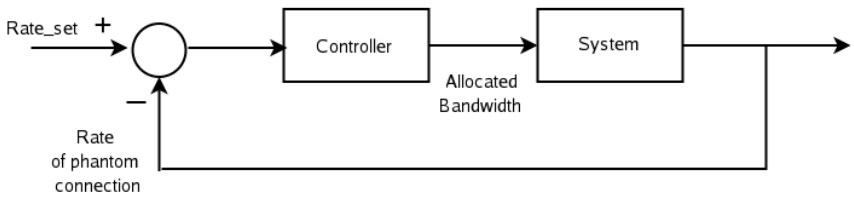

Figure 4.2: Control Loop

performance using closed-loop control. We propose to use reinforcement learning-based systematic methods to obtain suboptimal values for the control parameters. For this purpose, first RL simulations are performed using M/G/1-PS Model. So, this model will be explained and after that details of RL simulations will be given.

4.3.1 Modelling TCP flows with M/G/1-PS

Vast majority of internet traffic is transported over TCP, therefore the traffic is mostly elastic. Most internet applications such as FTP, HTTP and Telnet use TCP. TCP adapts its transfer rate to the current network status. By applying certain congestion avoidance mechanisms TCP achieves fair capacity sharing among active flows. TCP protocol uses packet losses as indications of congestion and responds to congestion by decreasing(typically halving) its rate. If no packet losses is observed, TCP increases its sending rate. Assuming that the TCP feedback mechanism is ideal and using these properties of TCP, a network link carrying only TCP traffic can be modelled as an M/G/1 Processor Sharing(PS) queue. In the M/G/1- PS Model, all active flows in the system get a fair share of the capacity. All active flows are served on a round robin fashion whereas in each round of service, each active flow receives a fixed quantum of service. Our reason of using this model is using a simple model for practical purposes. However, this model is not always a very accurate model and has some limitations since it does not take queueing and RTT into account. A useful measure of performance

which is the flow throughput(γ) is given by the formula $\gamma = \frac{E[\sigma]}{E[R]}$, where σ is the flow size and R is the response time of an arbitrary flow.

In M/G/1-PS Model that we use, flows arrive according to a certain distribution with rate λ flows/sec. The arrival process may be approximated as Poisson when the number of sources are large. The mean flow size is σ bits. Link capacity is C bits/sec. Flow size distribution has a heavy-tailed nature which means that most flows are very small (mice) but majority of traffic is contained in very long flows (elephants). According to M/G/1-PS Model, distribution of number of flows in progress is geometric. Main performance measure for elastic traffic carried by TCP is in terms of throughput and delay. In this model, the expected response time of a flow of size s can be calculated as $R(s) = E[\text{response time}] = \frac{s}{C(1-\rho)}$.

4.3.2 Reinforcement Learning Formulation

A controller with a feedback loop from the network will be used to set the average rate that the TCP flows get to the desired level, say $r_{desired}$. The feedback from the network is the measurement result of the average rate that the flows achieved in the last period of measurement (T). Using the weighted result of this measurement with the previous measurements, the feedback value at time n (r_n) is calculated. The controller determines the new value of the capacity (C_{n+1}) that will be allocated to the TCP flows so as to obtain the desired average rate $r_{desired}$. The desired value of the average rate should be satisfied with minimal variance from mean ($r_{desired}$) while using the capacity resource efficiently. In order to determine the parameters of the controller the system with uncertainty should be modelled using a discrete approach and solved to find the optimal values of the parameters.

4.3.3 SMDP Formulation

The problem is formulated as a Semi Markov Decision Process (SMDP). The feedback from the network which is the rate of the infinite phantom connection (r) is the state of the system. The rate values are discretized in order to obtain a small finite number of states. The actions are the discretized capacity values (C) allocated to the link. In MDP formulation, there is an associated probability of going from state i to state j under a certain action a, denoted as $p(i, a, j)$. According to the current state, the learning agent decides to select the next action so as to minimize the long run average cost. The immediate cost (C_{inc}) is defined as the error (e), which is the squared difference between the desired rate (r_{set}) and the current rate (r), i.e., $e = (r_{set} - r)^2$ multiplied by a constant. According to the feedback from the network about the current state of the system, the bandwidth allocated to the system is updated by taking the next action accordingly. The value of cost function determines how close the current action is to the optimal one that meets the rate constraint. If the current state is closer to the desired one , the cost is smaller. Gosavi's Reinforcement Learning algorithm explained in the previous chapter will be used to obtain a suboptimal policy $\pi(s)$. Using this policy, the decision maker chooses the action to take in each state. The simulations are performed to test the resultant rates obtained by the policy.

4.3.4 Gosavi's RL algorithm to solve SMDP

The Gosavi's Reinforcement Learning algorithm is used to learn a suboptimal policy. The general learning scheme can be summarized as follows: Let the system be in state s_t at time step t. The learning agent chooses an action a and receives an immediate feedback in terms of an immediate cost or reward. This action and the arrival of new flows change the network status and the system moves to a new state s_{t+1}. The learning agent selects another action from this

new state. This can be an either minimum cost action or an exploratory action to discover states that are not yet visited or visited a few times. If minimum cost action is chosen, the Q factors corresponding to the chosen action and the previous state are updated according to the rules of learning algorithm based on Bellman equation and using the values of the average cost and minimum cost from that state. The performance of an action in a state serves as an experience to the learning agent which is used to update its knowledge. The current choice of an action effects the following states, therefore a *return* concept is needed. Hence, picking action a at state s will give an total return $R(s,a)$, which is the weighted sum of future rewards from that state. In RL, the aim is to minimize the expected return $Q(s,a)$ given in terms of costs (or maximize $Q(s,a)$ if it is in terms of rewards). Thus, the performance metric that we use in the simulation is average cost (ρ) calculated by summing the immediate rewards earned and dividing this sum by the number of transitions. Since the number of states and actions are finite and small in our simulations, we keep a matrix for $Q(s,a)$ values. Depending on the choice of performing either exploration or exploitation, either a random exploratory less visited action is chosen or the action with minimum cost is chosen from this matrix.

Gosavi's RL algorithm that is explained in the previous chapter needs some learning parameters to be set to proper values. The learning parameters α and β are initialized to 0.5 and updated during the simulations. α is the averaging parameter for updating the cost ($Q(s,a)$) for a certain state action pair, and determines the ratio of the old value of $Q(s,a)$ that will be used to calculate the new value. β is the averaging parameter for updating the average cost ρ. The parameter α is updated using the number of visits of that state action pair ($visit(s,a)$) as follows: $alpha = 0.5/(int(visit(s,a)/100) + 1)$, which means α is halved for every 100 visits to that state. At each state either an action with minimum $Q(s,a)$ value is chosen (exploitation) or a random exploratory action is performed. A constant p is randomly chosen between (0,1)

and if $p \leq p_{explore}$, exploitation is performed, otherwise exploration is performed. The parameter $p_{explore}$ is updated according to the current time step n and the maximum number of steps in the simulation, which is 100 million as follows: $p_{explore} = 0.9/int(steps/(0.1 * MAX\_STEPS)) + 1)$. The parameter β is also updated using the same formula as $p_{explore}$.

4.3.5 RL Simulation Results

We modelled the system by an M/G/1-PS model where the elastic flows arrive according to a Poisson process with mean interarrival time T_f and have the following distribution given in [14]. 90% of the flows are mice with size uniformly distributed between 1 and 9 Kbytes; 10% of the packets are elephants with size uniformly distributed between 10 Kbytes and 400 Kbytes. The mean flow volume in Kbytes is denoted by β which is 25 Kbytes for this example. We use a infinite phantom connection to monitor the throughput of an arbitrary active flow.

Long-run RL simulations are performed for different values of the mean flow interarrival time T_f and a suboptimal policy is found for each T_f. The RL simulations are executed 100 million steps so that each state is visited large number of times and each state action pair is performed a large number of times as well. We then apply a linear regression on the obtained policies so as to determine a proportional control law in the form $u = K_p e + D$ for each T_f by calculating the parameters of the proportional controller K_p and D. However, evaluation of suboptimal policy show that, using linear regression includes additional error to the policy so it is better to keep a look up table of policies for each state action pair.

The traffic model is summarized in Table 4.5. For different values of arrival rates the RL simulations are performed. In addition RL simulations are also performed for different value of desired average throughput(T_{set}) being 200 Kb/s and

500 Kb/s. The resultant suboptimal policies obtained using Gosavi's Algorithm are given in Tables 4.3 and 4.2.

Traffic Model		
Arrival rate (flows/s)	$E[T_f](ms)$	ρ
$\lambda_1 = 157.48$	6.35	0.7
$\lambda_2 = 135$	7.41	0.6
$\lambda_3 = 112.49$	8.89	0.5
$\lambda_4 = 90$	11.1	0.4
$\lambda_5 = 67.5$	14.81	0.3
C_{max}	45 Mb/s	
β	25 packets	
packet size	1 Kbyte	

Table 4.1: *Traffic Model: The arrival rates, mean flow size β, maximum available capacity C_{max} and loads calculated using these values wrt C_{max} are given.*

Suboptimal Policies Obtained Through RL Simulations $T_{set} = 500Kb/s$					
States(rate)(Kb/s)	Actions(C)(Mb/s)				
Arrival Rate (Flows/s)	$\lambda_1 = 157.48$	$\lambda_2 = 135$	$\lambda_3 = 112.49$	$\lambda_4 = 90$	$\lambda_5 = 67.5$
$0(rate < 100)$	45	43	38	28	24
$1(100 \leq rate < 200)$	45	43	38	27	23
$2(200 \leq rate < 300)$	45	41	38	26	22
$3(300 \leq rate < 400)$	41	39	36	26	21
$4(400 \leq rate < 450)$	41	37	34	25	20
$5(450 \leq rate < 470)$	39	35	33	24	19
$6(470 \leq rate < 480)$	38	33	30	24	18
$7(480 \leq rate < 490)$	37	32	28	24	17
$8(490 \leq rate < 495)$	36	31	27.5	23	16
$9(495 \leq rate < 500)$	36	30	27	23	15
$10(500 \leq rate < 505)$	36	30	26.5	23	15
$11(505 \leq rate < 510)$	36	30	26	23	15
$12(510 \leq rate < 520)$	35	29.5	25.5	22	14
$13(520 \leq rate < 530)$	35	29	25	21	13
$14(530 \leq rate < 550)$	34	28	24	20	13
$15(550 \leq rate < 600)$	31	27	23	19	12
$16(600 \leq rate < 800)$	30	27	23	18	12
$17(rate \geq 800)$	30	26	22	17	12

Table 4.2: *Look up Table Versions of Suboptimal Policies Obtained Through RL Simulations for $T_{set} = 500Kb/s$ using M/G/1-PS Model of the TCP simulated in C++ for different arrival rates*

The policies are fitted to a line using linear regression and the parameters of the controller are determined from this line. So the proportional controller may be applied for a certain capacity ranges. However, if the limits of capacity are exceeded, if the controller calculates the capacity larger than maximum capacity in the policy, capacity is set to the upper limit, if it is smaller than minimum

Suboptimal Policies Obtained Through RL Simulations $T_{set} = 200Kb/s$				
States(rate)(Kb/s)	Actions(C)(Mb/s)			
Arrival Rate (Flows/s)	$\lambda_1 = 157.48$	$\lambda_2 = 112.49$	$\lambda_3 = 90$	$\lambda_4 = 67.5$
$0(rate < 80$)	39	28	28	21
$1(80 \leq rate < 110$)	39	27.5	27	21
$2(110 \leq rate < 130$)	37	27	26	20
$3(130 \leq rate < 150$)	37	27	25	19
$4(150 \leq rate < 160$)	37	27	24	18
$5(160 \leq rate < 170$)	37	26	23	17
$6(170 \leq rate < 180$)	37	26	22	16
$7(180 \leq rate < 190$)	35	26	21	15
$8(190 \leq rate < 195$)	35	25	20.5	14
$9(195 \leq rate < 200$)	34	24.5	20	14
$10(200 \leq rate < 205$)	33.5	24	19.5	13.5
$11(205 \leq rate < 210$)	33	23	19	13
$12(210 \leq rate < 220$)	32	22.5	18.5	13
$13(220 \leq rate < 230$)	30.5	22	18	12
$14(230 \leq rate < 250$)	30	21	17.5	12
$15(250 \leq rate < 300$)	29	21	17	11
$16(300 \leq rate < 500$)	29	20	16	11
$17(rate \geq 500)$	29	20	15	10

Table 4.3: *Look up Table Versions of Suboptimal Policies Obtained Through RL Simulations for $T_{set} = 200Kb/s$ using M/G/1-PS Model of the TCP simulated in C++ for different arrival rates*

value it is set to the lower limit. However, using the suboptimal policies in a lookup table form gives better results that are close to the desired rate. Therefore the experiments are performed using a look up table version. In addition, as a future work, different controllers can be used to improve the results, such as derivative controller. This controller uses the difference between current rate and the previous rate as input and calculates the output by multiplying this difference by a constant control parameter.

RL results: Proportional Controller Parameters				
T_{set} (Kb/s)	200		500	
Arrival Rate (Flows/s)	K_p	D	K_p	D
$\lambda_1 = 157.48$	-17.5	42259.5	-26.3	49549.1
$\lambda_2 = 112.49$	-13.1	29800	-29.6	43021.1
$\lambda_3 = 90$	-21.1	30865	-16.5	30711.5
$\lambda_4 = 67.5$	-19.2	24032.2	-21.2	26660.7

Table 4.4: *Proportional controller parameters determined by applying linear regression to the policies obtained for different arrival rates and T_{set} values.*

After the learning simulations are performed and the suboptimal policies are obtained, these policies are evaluated using both M/G/1-PS model of TCP and using ns-2 simulator [33]. The results are given in the following tables and figures. The rate of the flows are measured using a phantom connection. The rate versus time graphs for desired rate (T_{set}) value of 200 Kb/s and for 2 different arrival rates of 112.49 flows/s and 157.48 flows/s are shown in Fig. 4.3 and 4.4.

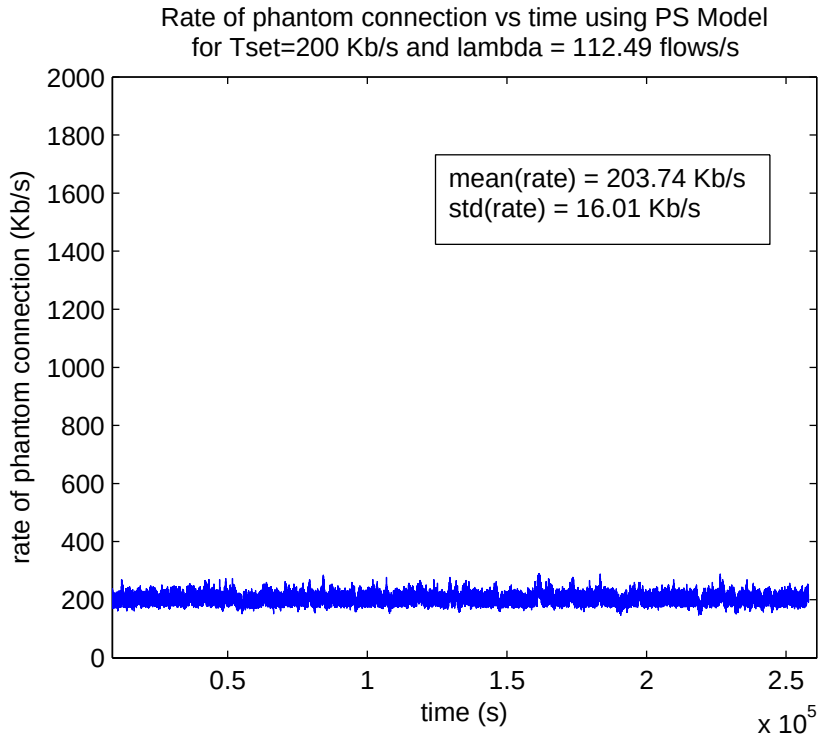

Figure 4.3: Evaluation of policies using PS Model: Rate of phantom connection vs time for $R_{set} = 200Kb/s$ for $\lambda = 112.49$ flows/s .

The results of the linear regression performed on the policies obtained for 2 different values of R_{set} and for different arrival rates are shown in Fig. 4.5 and 4.6.

The results of performing the obtained policies using M/G/1-PS model are tabulated in Table 4.5. The dynamic link provisioning results are compared with the static case. The average of capacity used (C_{avg}) in the dynamic link provisioning simulations are calculated from the recorded capacity values.

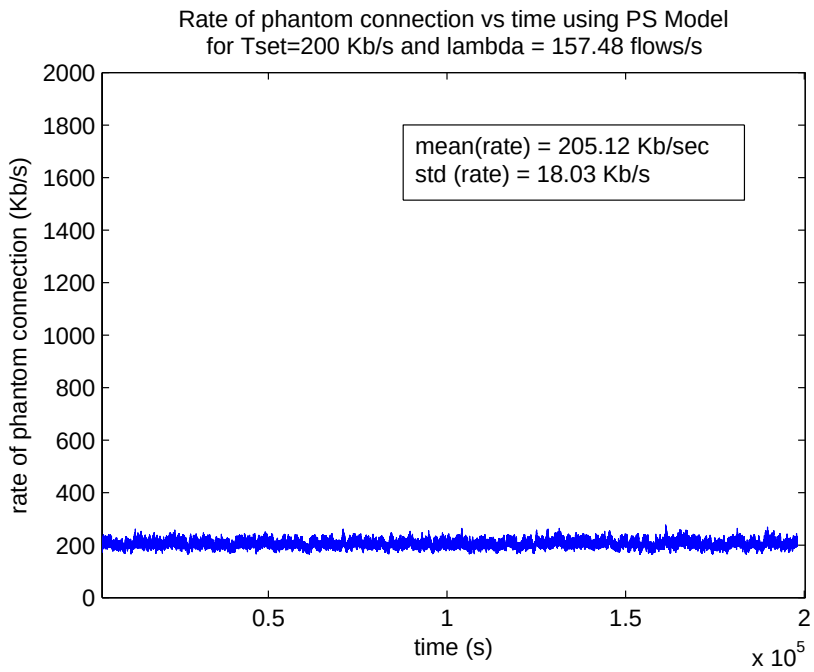

Figure 4.4: Evaluation of policies using PS Model: Rate of phantom connection vs time for $R_{set} = 200Kb/s$ for $\lambda = 157.48$ flows/s .

The suboptimal policies obtained from the RL simulations are also tested using ns. According to the feedback from the network, which is obtained by monitoring the rate that the infinite phantom connection gets in the last time interval, the bandwidth required is found from the look up table of the obtained policy. However since this rate values fluctuate, exponentially weighted moving average of previous rate and the current rate is calculated using the equation: $\beta r_{current} + (1 - \beta) r_{prev}$, where β is taken to be 0.6 in these simulations. The dynamic link provisioning simulations in ns results in mean rates values which are close to the desired mean rates. However, standard deviations are much larger in ns. The results are given in Table 4.6.

The results of dynamic link provisioning are compared with the static case, and it is observed that dynamic link provisioning results in much smaller standard deviations from the desired mean when compared with the static case. In static provisioning simulation, the capacity of the link is chosen based on the

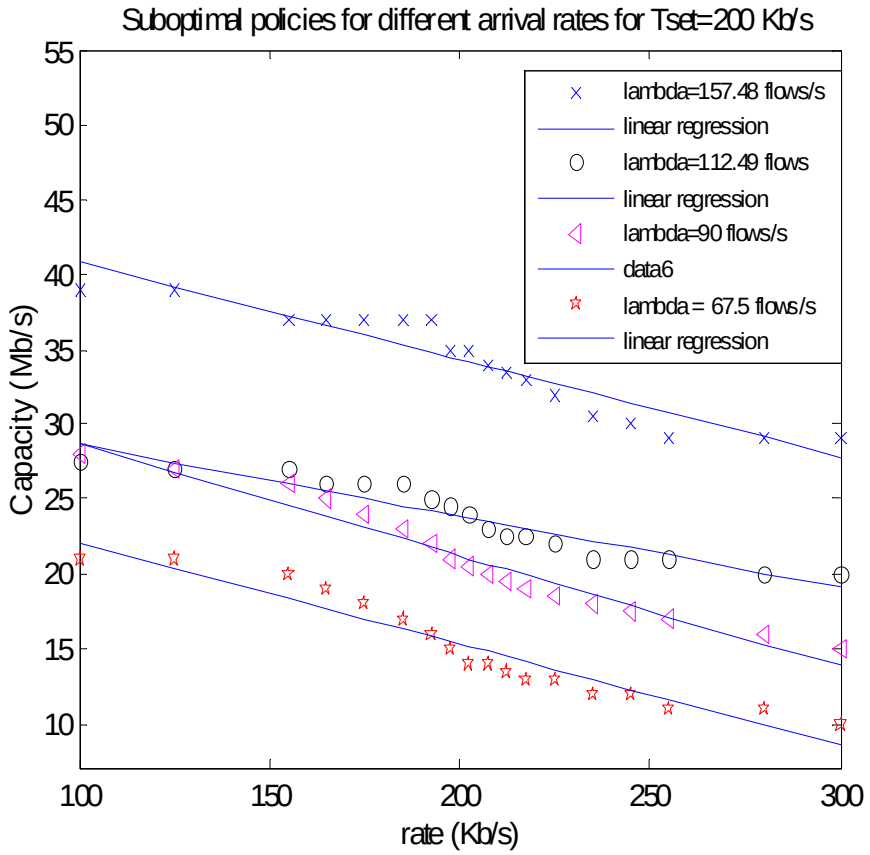

Figure 4.5: Suboptimal Policies for $R_{set} = 200Kb/s$ for different λ's

mean values of the capacities calculated from the dynamic provisioning simulation results. In addition, we tried to obtain $T_{set} = 200Kb/s$ in average in static case to compare the results with the static case. This value cannot be obtained through processor sharing (PS) formulas since there is an additional infinite phantom connection, the effect of which is not included in the PS formulas. Resultantly, it is observed that, rate in the static simulations oscillates too much and takes very small values in case burst of flows arrive.

The rate versus time plots for the phantom connection is plotted in Fig. 4.7 and 4.8 for two different arrival rates and T_{set} values using their corresponding policies obtained previously. The values are calculated by dividing the total amount of bytes received by the receiver divided by the time interval of measurement. Here time interval is 0.4 sec.

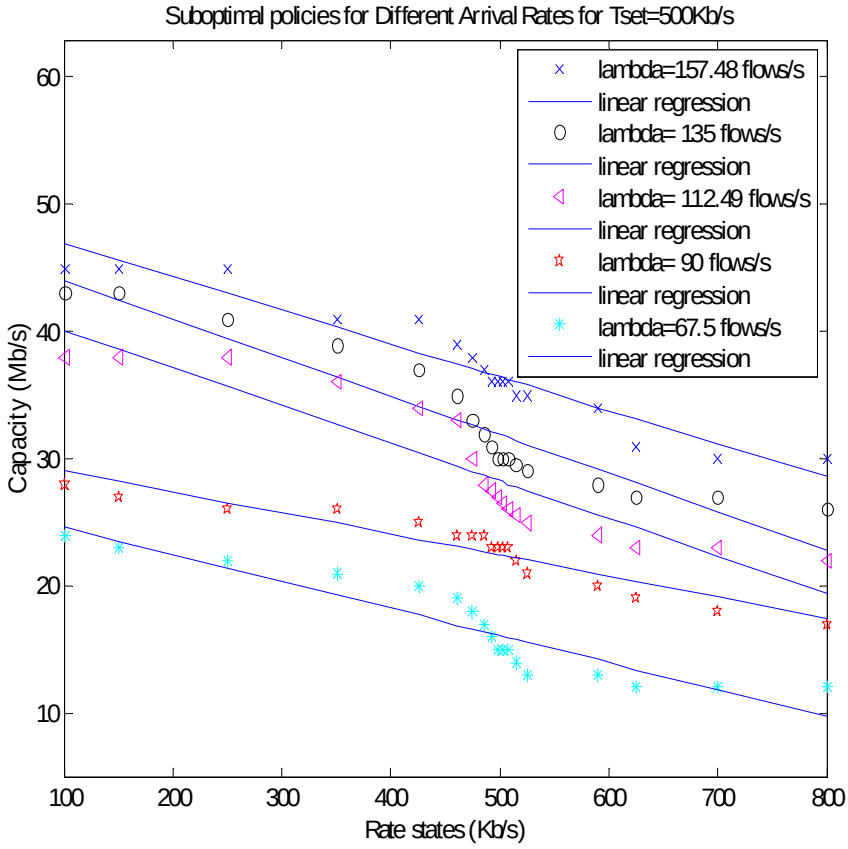

Figure 4.6: Suboptimal Policies for $R_{set} = 200Kb/s$ for different λ's

PS Model Results (Dynamic Link Provisioning)				
R_{set} (Kb/s)	200		500	
Arrival Rate (Flows/s)	$\lambda_1 = 157.48$	$\lambda_2 = 112.49$	$\lambda_1 = 157.48$	$\lambda_2 = 112.49$
Mean(T) (Kb/s)	205.11	203.74	469.18	475.29
$\sigma(T)$ (Kb/s)	18.02	16.01	42.35	62.62
Mean(C) (Mb/s)	33.186	23.824	38.336	29.649

Table 4.5: *Dynamic Link Provisioning: The results of suboptimal policies obtained through RL simulations and evaluated using M/G/1-PS Model of the TCP simulated in C++ for two different arrival rates and two different desired mean rate values (R_{set})*

NS Results (Dynamic Link Provisioning)				
T_{set} (Kb/s)	200		500	
Arrival Rate (Flows/s)	$\lambda_1 = 157.48$	$\lambda_2 = 112.49$	$\lambda_1 = 157.48$	$\lambda_2 = 112.49$
Mean(T) (Kb/s)	199.19	201.05	498.04	516.47
$\sigma(T)$ (Kb/s)	112.20	131.53	181.80	121.02
Mean(C) (Mb/s)	33.691	24.235	33.948	24.463

Table 4.6: *Dynamic Link Provisioning: The results of suboptimal policies obtained through RL simulations and evaluated using NS for two different arrival rates and two different desired mean rate values (T_{set})*

NS Results (Static Link Provisioning)						
Arrival Rate (Flows/s)	$\lambda_1 = 157.48$			$\lambda_2 = 112.49$		
constant C (Mb/s)	33.400	33.500	33.700	24.000	24.200	24.280
Mean(T) (Kb/s)	79.61	167.99	201.10	98.11	184.92	195.91
$\sigma(T)$ (Kb/s)	63.60	138.56	164.51	171.67	174.55	181.13

Table 4.7: *Static Link Provisioning: The static link provisioning simulated using NS for two different arrival rates and for desired mean rate value (T_{set}) being 200 Kb/s*

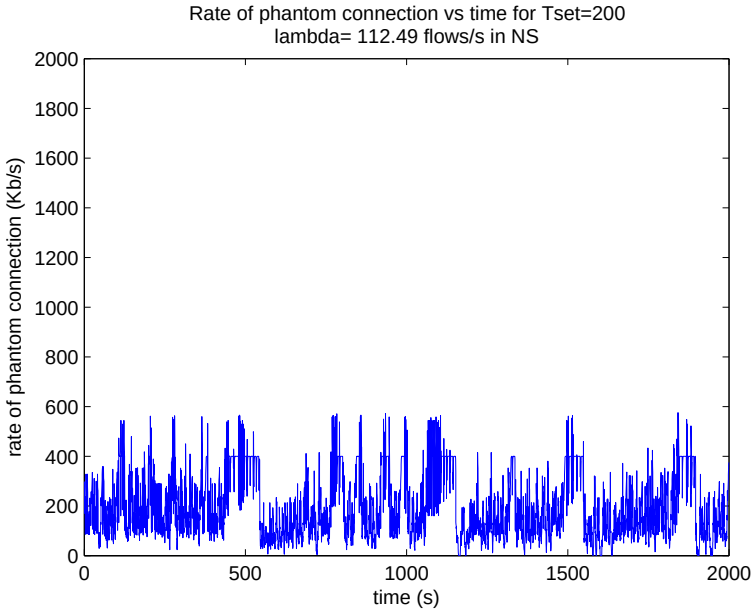

Figure 4.7: Rate of phantom connection vs time for $T_{set} = 200Kb/s$ for $\lambda = 112.49 flows/s$ in NS simulations.

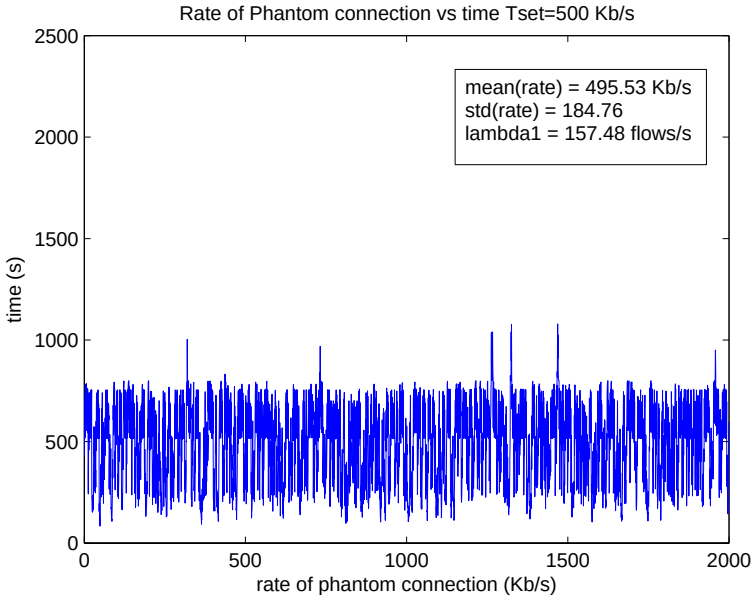

Figure 4.8: Rate of phantom connection vs time for $T_{set} = 500Kb/s$ for $\lambda = 157.48 flows/s$ in NS simulations.

Chapter 5

Dynamic Selection of Weights in Deficit Round Robin Scheduling

5.1 Related Work

To support multiple classes with different service requirements, different approaches were used in the literature. As explained in chapter 2, DiffServ architecture is used for serving different QoS requirements of a small number of classes. In the recent literature as opposed to the more complex QoS models such as IntServ and DiffServ, simpler QoS mechanism such as flow aware networking are proposed [34] [22]. These traffic control and QoS schemes are local and independent from the other parts of the network. The user defined flows are identified from their packet headers and the QoS mechanisms such as admission control and scheduling are applied to these flows at the individual routers independently.

At the core routers, the key problem is to decide how to allocate resources to each class to satisfy their QoS requirements. In this context, different classes usually belong to different applications like video, voice, data that have different

QoS requirements. Most of the present literature prioritize certain traffic types such as delay and loss sensitive traffic. So streaming traffic applications have higher priority then data traffic. However, there is also a need to satisfy different QoS requirements of elastic traffic. The most common measure of QoS for elastic traffic is the throughput. Thus, using this measure different QoS requirements of different classes of users need to be satisfied.

In order to give different treatment to different classes of traffic according to their requirements, each class of traffic is inserted into a different queue and a scheduler decides which queue to serve next and what portion of capacity is needed to be allocated to that queue. The second method to control traffic and satisfy QoS at the routers is to control queue length [8] using different schemes such as Random Early Detection (RED) [13].

There are various scheduling algorithms proposed in literature. Some of these are explained in the scheduling section of Chapter 2 like FIFO, Fair Queueing, Stochastic Fair Queueing, Weighted Fair Queueing(WFQ). FIFO is the simplest scheduler which does not implement different treatment of classes. Some of these schedulers are fair but computationally expensive. For example, WFQ uses the packet slice concept and precisely guarantees bandwidth and delay. However, a large amount of packet processing is required since packet sorting and other mechanisms are used in WFQ. Therefore, these schedulers are not suitable for high speed links. In Priority Queueing (PQ), the lower priority queue is served only if all queues that are of higher priority have already been served. This causes lower priority classes to starve. In Weighted Round Robin (WRR) algorithm and some different versions of it, fairness is traded off for computational speed. In this scheduler, the determination of weights is an important issue. In the WRR scheme, at each round, the scheduler dequeues and transmits a number of packets from each queue that is equal to its weight. However, these algorithms require exact knowledge of packet lengths, since the amount of bandwidth allocated

for each class depends on packet lengths. Otherwise fairness between different classes depends on packet length and WRR scheme becomes unfair when packet lengths are different. Deficit Round Robin (DRR) Scheduler is used to solve this problem, and it will be explained in details in the following sections of this chapter.

In order to solve the fairness problem of WRR scheduler, Variably Weighted Round Robin(VWRR) Scheduler is proposed in [18]. The weights of the VWRR scheduler are updated adaptively depending on the average packet length. The average packet lengths of the flows of each queue (p_i) are calculated. Then, maximum of these packet lengths are found (p_{max}). The weights required by each queue (w_i) are first determined according to their bandwidth requirements, then these weights are scaled by a ratio given by p_{max}/p_i to take the packet length into account. The amount of fairness achieved depends on measurement intervals.

When bursty traffic arrives, WRR scheme causes further delays. There are several different proposed schemes to deal with bursty traffic by updating weights according to the network status. However, in some of these schemes, only queue length is considered to update weights. Therefore, the lower priority classes also receive large weights, which is undesirable in the DiffServ architecture, since fairness among different priority classes is not preserved. However, in Fair Weighted Round Robin Scheme (FWRR) [26] the weight of each service class changes but the fairness among different priority classes are preserved. In FWRR, an ordinary WRR scheme is used when the network is not congested. When network is congested and there are significant bursts of data, the queue length increases. When the queue length of one of the priority classes exceeds a predefined threshold, the weight update mechanism starts. The weight of the congested queue is increased by a fraction $\beta \in (0, 1)$ and all the other priority classed update their weight by the same amount to preserve the fairness among queues. When queue

length of priority queues drop below the threshold queue weights are changed back to their original values. This mechanism preserve fairness among queues.

The problem with different schedulers like WRR or PQ is that, the performance is highly dependent on control parameters. Optimal choice of these parameters is difficult. In [20], a Learning Based Scheduling algorithm is proposed. Their objective is to schedule packets with the aim of providing sufficient service to each traffic stream so that their objectives are just satisfied, thus maximizing resources left for other streams. In other words, the delay of the best effort traffic is minimized while satisfying QoS goals for upper classes. A learning automaton which has finite number of actions is used. The automaton, chooses an action, which is choosing the queue to be serviced in the next interval. So each action corresponds to one of the queues. Each action has a probability of being chosen and the probability of selecting a queue is updated after p packets are served from that queue. The probability of selecting a queue is increased if its performance is less than the desired performance. If its performance is better than desired, the opposite is done. Different performance measures may be used but this work uses mean delay. M_i is the measured delay and R_i is the desired delay. If the performance is less (or greater) than desired the probability of choosing that queue is increased (or decreased) by an amount $\beta_i a$, where β_i is the difference between M_i and R_i divided by a normalization constant and a is a constant. This performance update is the feedback from the network for the action chosen in the form of an immediate reward. The limitations of this algorithm are that, this algorithm still depends on the choice of certain parameters and the scheduling policy does not take into account the current state of the system such as the state of each queue, when choosing an action.

The weights of the schedulers can be set statically to satisfy a worst case delay bound, however, this causes fixed allocation of bandwidth and when higher priority classes are not utilizing their bandwidth, best effort traffic still starves

from resources. In [1], Reinforcement Learning is used to learn a scheduling policy in response to the feedback from the network about the delay experienced. According to the changing traffic conditions, the scheduler is updated. In addition to [20], this paper uses the system state information as a feedback from the network to update the probabilities of each queue. In the Reinforcement learning formulation, the states, the reward(or cost) function and the learning algorithm must be determined first. The states are defined as binary variables corresponding to each queue q_i and take value 1 if the that queue is meeting its mean delay requirement or take 0 else. The reward function aims to provide a positive feedback when the delay requirements are met or a negative feedback else, and this reward is proportional to the value given by measured delay divided by the required delay . The Q-learning algorithm [38] is used to learn the scheduling policy. The results show that the requirements of the queues are satisfied while not starving the best effort traffic.

In the recent literature there are many proposed approaches to provide QoS through different scheduling algorithms based on DiffServ architecture. Some of these are explained above. However, there is another approach called Relative Differentiated Services (RDS) which is proposed to fill the gap between best effort service and DiffServ. In RDS model, the admission control and resource reservation tasks are not performed. Hence, RDS model does not provide absolute service guarantee but packets with higher priority will receive better service than that of low priority. There are several different schedulers proposed to support the RDS model [25]. In [12] , some former methods based on time stamping each arriving packet and computing the waiting time of head-of-line packets, then determining the scheduling priority starting from the smallest time stamped ones are explained. However, [25] argues that serving the packet with the smallest time step leads to bottlenecks. Hence, many other approaches are proposed that depend on monitoring the arrival rates and queue lengths [25], [29], [43] ,[39], [41]. In these approaches, the service rates or parameters to determine priorities

are updated at each interval depending on the feedback obtained from network. One other issue for prioritizing the traffic is considering the packet lengths. Serving the shortest packets first policy decreases the queueing delay, especially for VoIP traffic that includes many small packets with delay requirements. However, all these approaches emphasize the delay differentiation between different classes of delay sensitive traffic. In our case, we want to differentiate TCP flows with different QoS requirements using a dynamic version of DRR scheduler, which will be explained next.

5.2 Deficit Round Robin Scheduler (DRR)

The mostly used queueing mechanism in routers is FIFO queue. In a FIFO queue, packets are queued on the order of their arrivals. Then, congestion control is implemented by the source, which decreases sending rate in case of congestion. However an ill behaved flow can keep increasing its share causing other flows to suffer. This is the reason of using separate queues for flows coming from different source destination pairs and serving these queues in a round robin fashion. The router prevents the ill behaved source from increasing its share of bandwidth because its queue length will increase and packets will be dropped. However, this mechanism ignores packet lengths, therefore a flow can get max/min times the bandwidth of other flow, where max is the maximum, min is the minimum packet size. Fair Queueing (FQ) is proposed as a solution to this problem, that is sending one bit at a time in a round robin fashion. This is not possible to implement so the time that the packet would left the router if this scheme were implemented is calculated. The packets are then inserted into the queue of packets sorted based on their time of departure. This sorting algorithm takes O(log(n)) times where n is the number of flows. This computational loads causes this scheme to be hard to implement at high speeds. Hence, Stochastic Fair Queueing Scheme is proposed to decrease the computations for the enqueue process to constant time (O(1))

by using hashing to map the packets to the corresponding queues. In addition, since, using a separate queue for each flow with same source destination pair causes the number of queues to be very large, in SFQ, less number of queues then the number of flows are used and hashing function determines the queue that each flow will be assigned. However, this causes some flows collide with others and colliding flows will be treated unfairly. Therefore fairness guarantees are probabilistic.

DRR [32] is most popular due to its low O(1) complexity compared with other high complexity fair queuing schemes. The flows arriving at the router are queued and wait to be enqueued to their corresponding queue according to their source destination pair based on the hashing function. DRR is proposed to overcome the fairness problem of WRR. DRR can guarantee fairness in terms of throughput. In DRR, a state variable called *Deficit Counter* is used to control the amount of bytes that is not served in that round due to the fact that packet length is larger than the bytes that is needed to be served from that queue at that round. The amount of bytes served from each queue is kept in a variable called *Quantum*. Hence, at each round *Quantum* is added to the value in *Deficit Counter* and for each queue, the amount of bytes to be served in the next round is calculated. If the length of the packet at the head of the queue is smaller than this value, that packet is dequeued and transmitted. The length of that packet is subtracted from the *Deficit Counter*. Otherwise, the packet is not served and the turn passes to the next queue. This mechanism is demonstrated in the Fig. 5.1 taken from the original paper [32]. In this example, there are three queues served on a RR fashion. Initially all *Deficit Counter* variables for each queue are set to zero and round robin pointer points to the top of *Active List*, which is list of queues that are nonempty. The *Quantum* is set to 500, so this value is added to *Deficit Counter*. After serving the head of line packet of size 200, the remainder 300 is left in the *Deficit Counter* since the next packet in line is larger than 300. Then the remaining queues are served similarly.

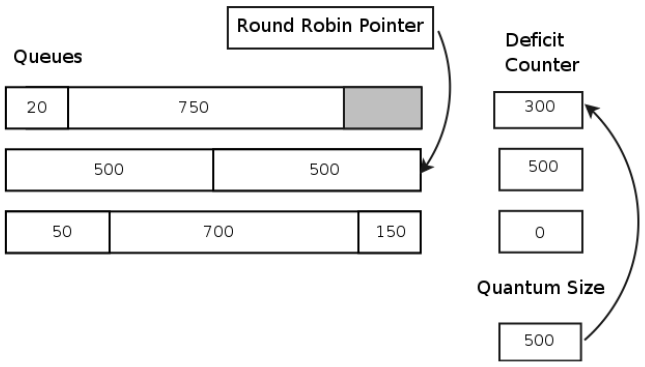

Figure 5.1: Deficit Round Robin Scheduling

The DRR queuing mechanism is also used to differentiate traffic with different service requirements. In DiffServ architecture, the packets are classified at the edge nodes and different priorities of traffic are inserted into different queues based on their DS field in their IP header. Each priority class have different service requirements. In most applications delay sensitive traffic is prioritized according to it minimum delay requirement. At the core nodes DRR scheduling can be used to serve these different priority queues. The latency and fairness of DRR queue are not optimal, therefore there are variants of DRR queue to satisfy different requirements of various applications. For example, Dynamic DRR by Yamokoshi et. al. [42] provide delay differentiation according to the packet lengths, resulting in shorter delays for smaller packets. This is achieved by changing the granularity (*Quantum*) of *Deficit Counters* instead of using fixed granularity. At each round, firstly the head-of-line packet with smallest size is served, then the granularity is set to its size, and then the queue with the minimum value of difference between head-of-line packet length and the value of its deficit counter is served. After that, the deficit counter of the unserved queues are increased by this difference.

Most of the literature concentrates on the QoS requirements of loss and delay sensitive traffic. In our work, we emphasize the QoS requirement of elastic traffic, which is measured in terms of the average throughput that TCP flows

receive. The traffic is separated into different queues with different QoS requirements. There is also a best effort queue in addition to these service classes. The bandwidth is allocated to these queues according to their requirements. Static allocation of bandwidths to each queue causes the best effort traffic to starve even when the higher priority classes do not utilize their allocated capacity. Therefore, a dynamic version of Deficit Round Robin Scheduler is used with the aim of allocating the smallest capacity to just satisfy the requirements of higher priority queues and the rest of the capacity will be used by the rest of the traffic which is taken to be best effort traffic in our scenario.

5.3 Dynamic Deficit Round Robin Scheduler (DDRR)

Consider an output link of a given router that implements Dynamic Deficit Round Robin(DDRR) Scheduling. There are n queues, that are numbered from 1 to n and served in a round robin fashion as shown in Fig. 5.2. Queue i is the ith queue which stores packets with flow id i. There is an associated *Deficit Counter* i for each queue i that stores the amount of bytes that queue i did not use in the previous round. Initially all *Deficit Counters* are set to 0. In addition, for each queue i, there is a *Quantum* i in terms of bytes which shows the amount of capacity that each queue can use at each round of service. In order not to keep examining empty queues, there is an auxiliary list called the *Active List*, which is a list of indices of queues that contain at least one packet. Each queue i from 1 to $n-1$ has a different QoS requirement in terms of desired average throughput. A controller is used for each queue to monitor the current rate by calculating the rate of the infinite phantom connection sharing capacity fairly with other active flows and change the bandwidth allocated to that queue according to a previously learned policy.

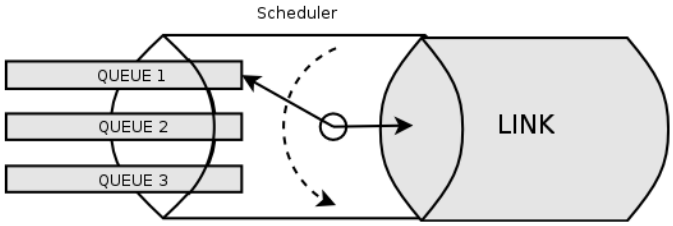

Figure 5.2: Round Robin Scheduling

The results of the introductory link provisioning problem are used for the DDRR Scheduling problem. In the previous chapter, learning simulations are performed for a single link to allocate the capacity dynamically so as to just satisfy the QoS needs in terms of the desired average throughput. A suboptimal policy for the controller is found using learning simulation for different arrival rates and different desired average throughput of flows. In DDRR, for each queue with different QoS requirements and arrival rates, the previously obtained policies will be used.

For each queue of the DDRR scheduler, there is an associated infinite phantom connection. Throughput of the flows are monitored by calculating the throughput of the infinite phantom connection at each interval t by dividing the amount of bytes of flow delivered to destination by the phantom connection to the time interval of measurement t. This is the performance measure for the TCP flows. According to the throughput of the phantom connection, which is continuously monitored, the resources are allocated to each queue according to the previously learned policy.

In our simulations, the system is composed of $n = 3$ queues, which are served in a round robin fashion. First queue serves class 1 traffic, which has a mean rate requirement of $T_{set}(1) = 500Kb/s$, while second queue serves class 2 traffic, which has a mean rate requirement of $T_{set}(2) = 200Kb/s$. Third queue is the best effort traffic which receives the capacity left from the other queues. Each

queue has an infinite phantom connection sharing capacity fairly with the flows of that queue, shown in Fig. 5.3. In addition, each queue also has a controller with a policy obtained previously using learning simulations. The aim of the controller is to keep the rate close to the desired mean rate which is $T_{set}(i)$ for queue i.

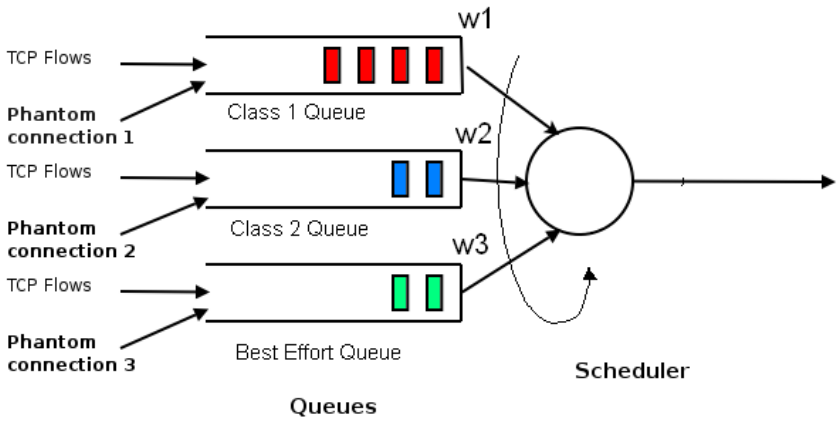

Figure 5.3: Round Robin Scheduling with Phantom Connection

The proposed DDRR algorithm shown in Fig. 5.4 can be summarized as follows: It is the same as DRR algorithm with only difference that *Quantum* of each queue are updated after each round in DDRR whereas the *Quantums* are constant in DRR. In order to update the *Quantum* of each queue, firstly, the capacity required for each queue called C_i is assigned by the controller of each queue. If the total capacity for queue 1 and queue 2 (Ctot=$C_1 + C_2$) is less than or equal to the total bandwidth of the link, say C Kb/s, then the *Quantums* of queue 1 and queue 2 are set in proportion to C_1 and C_2 and the remaining capacity is allocated to best effort traffic. For example if the required capacity is found to be 29000 Kb/s for queue 1 and 23000 Kb/s for queue 2, since total of two being 52000Kb/s, is smaller than C, say 65000Kb/s, the quantum of queue 1 will be set to 2900 bytes and that of queue 2 will be set to 2300 bytes and the remaining 1200 will be given to the best effort queue. If the total required capacity of first 2 queues are greater than C, a very small *Quantum* is assigned to the best effort queue and the remaining large amount of capacity is allocated

to class 1 and 2 queues in proportion to their requirements. However, in the second case, which is the overload situation, the amount of capacity allocated to each queue is less than the required capacity to satisfy its QoS constraint. Therefore, the desired rate cannot be obtained for the flows in this overload situation. However, still, the flow are expected to receive rate proportional to their required capacity according to the policy applied by the controller. The proposed DDRR algorithm is simulated in ns-2.

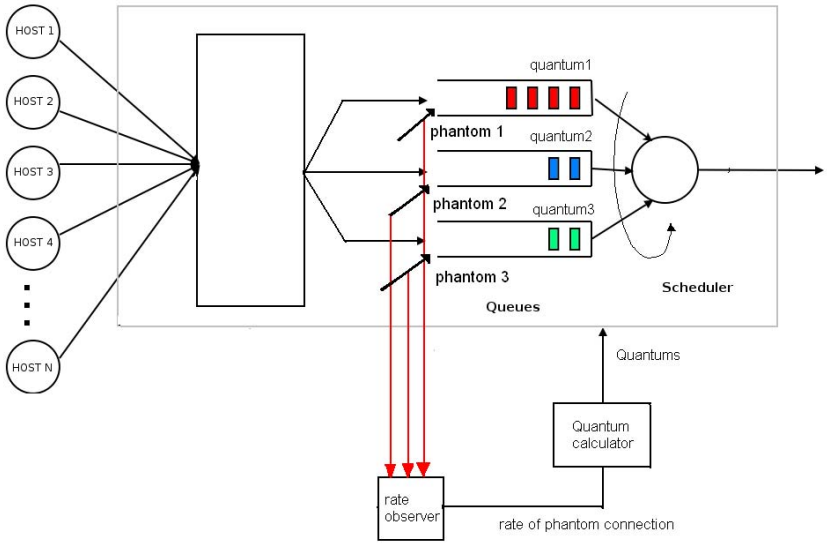

Figure 5.4: Dynamic DRR schematic

5.4 Simulation Scenario

There is a large number of hosts connected to a single router serving for an output link. Since the number of hosts is large, the arriving traffic can be modelled by a Poisson distribution. So from each host generating TCP flows, flows arrive with a total mean arrival rate of λ flows/sec. The size of the flows have the following distribution given in [14]. 90% of the flows are mice with size uniformly distributed between 1 and 9 Kbytes; 10% of the packets are elephants with size uniformly distributed between 10 Kbytes and 400 Kbytes. The mean flow volume in Kbytes is denoted by β which is 25 Kbytes in our simulations. The router

runs the DDRR algorithm. At the router, the flows are queued in the input buffer and wait for an *enqueue* action which finds the right queue for each flow according to its flow id. In the simulations, when the flows are generated, they are assigned a flow number and each flow is inserted in the appropriate queue according to this number. Therefore each packet arrives to each queue with a certain arrival rate $\lambda_{q(i)}$. There are two different queues with different average rate requirements, i.e, 500 Kbits/sec and 200 Kbits/sec and one more queue which is for best effort. According to the arrival rate for each queue, previously calculated policies are executed and the capacities allocated to each queue are updated during the simulation based on the feedback from the network and using the policies stored in the look-up tables.

5.5 Simulation Results

The proposed DDRR algorithm is tested using the ns-2 simulator by adding our proposed scheduler to the standard ns-2 distribution. The policies obtained for each queue and for different arrival rates using the learning algorithm described in the previous chapter are inserted to the scheduler and the weights of the scheduler are updated dynamically. Since our scheduler is a dynamic version of the DRR scheduler, instead of weights, which are in terms of number of packets to be served in each round, *Quantums*, which are in terms of bytes are used.

In the first simulation scenario, each three class of flow arrives to the queue with the same arrival rate, 112.49 flows/sec and they are enqueued their corresponding queues based on their flow numbers. The rate of the phantom connection is monitored. The policies obtained for the given arrival rate and for two different queues are tabulated in the previous chapter. In this simulation, the total capacity requirement of the first two queues C_{tot} is always smaller than the link capacity, which is set to $C = 65Kb/s$. The results of the simulation are

Queues	Queue 1	Queue 2	Queue 3 (Best Effort)
T_{set} (Kb/s)	500	200	N/A
Mean(T) (Kb/s)	508.88	195.39	45.19
$\sigma(T)$ (Kb/s)	84.88	88.83	40.11

Table 5.1: *Results Dynamic DRR Scheduler: Mean rates of phantom connection for each queue, the standard deviation of rates are given for arrival rate* $\lambda = 112.49 Flows/sec for each queue and total capacity of the link C = 65\ Mb/s$

tabulated in table 5.1. The mean rates for phantom connection for each queue and the standard deviation from the mean are given as well. The results show that, the policies obtained through learning simulations satisfy the average rate requirements for the phantom connection of each queue.

In addition to the tabulated statistics of phantom connection rate for each queue, the rate versus time behavior of phantom connection for queue 1 with a mean rate requirement of 500 Kb/s and queue 2 with a mean rate requirement of 200 Kb/s are given. The rate of phantom connection is recorded at each period, which is set to be 0.4 s in these simulations. The resultant figures are given in Fig. 5.5 and Fig. 5.6.

In addition to the rate of the phantom connection for each queue, which is controlled by the policy executed by the controller, the rate statistics of the individual flows are also recorded during ns simulations. One issue with the individual flows is that, throughput of small flows (mices) are severely limited by the slow start mechanism of the TCP. Throughput of large flows depend more on TCP congestion avoidance and tends to that of phantom connection. Therefore, the rate statistics of the flows whose length are greater than a certain value are recorded. This value is set to be 100 Kbytes as shown in Figs. 5.7 and 5.9 below. Since, for the second queue smaller flows may receive 200 Kbits/s, another histogram showing flows with length greater than 20 Kbytes is shown in Fig. 5.8. These flows whose length are greater than a certain amount , which is taken to be 100Kbytes in these simulations, are in bytes 97% of the total volume of the traffic transmitted.

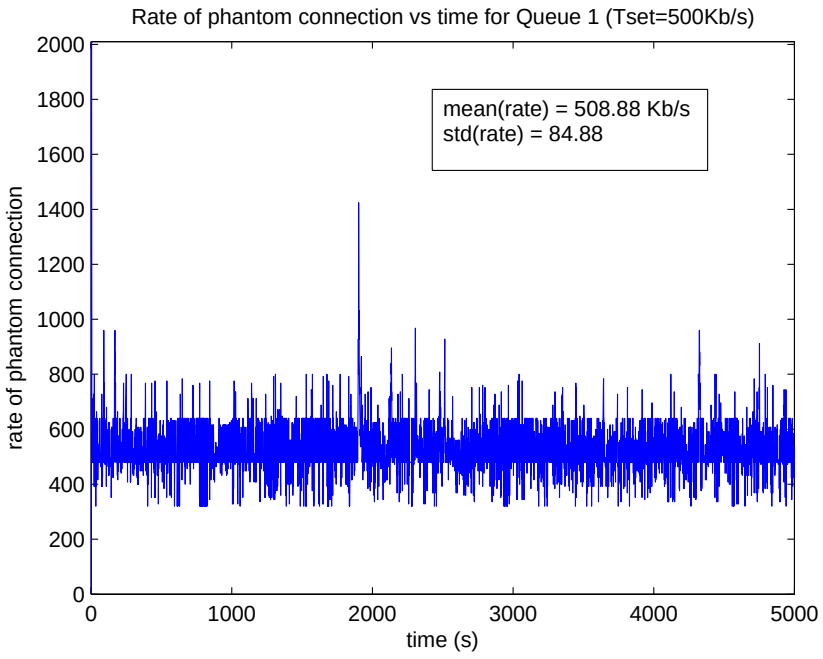

Figure 5.5: Rate of phantom connection vs time for queue 1 with $T_{set} = 500Kb/s$ for $\lambda = 112.49$ flows/s in Dynamic DRR ns simulations.

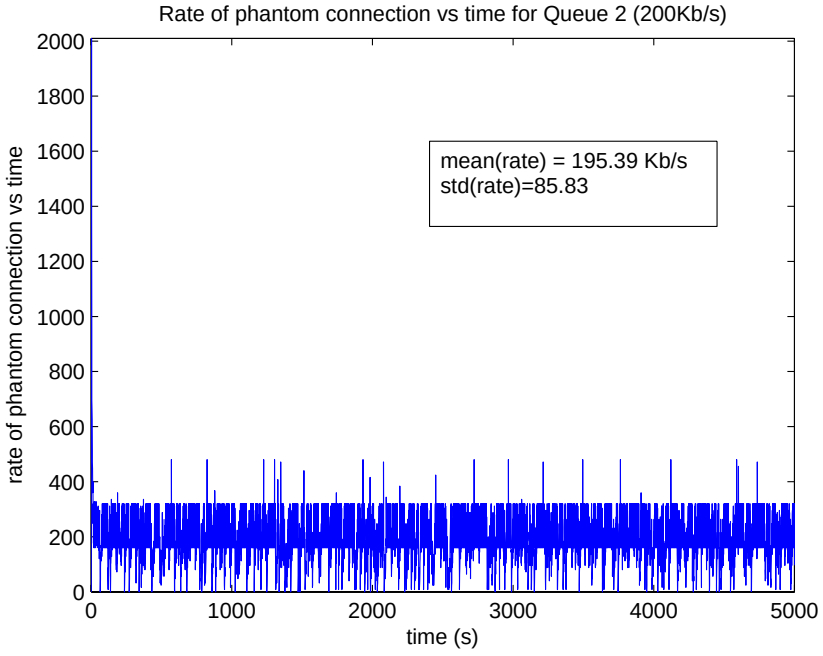

Figure 5.6: Rate of phantom connection vs time for queue 2 with $T_{set} = 200Kb/s$ for $\lambda = 112.49$ flows/s in Dynamic DRR ns simulations.

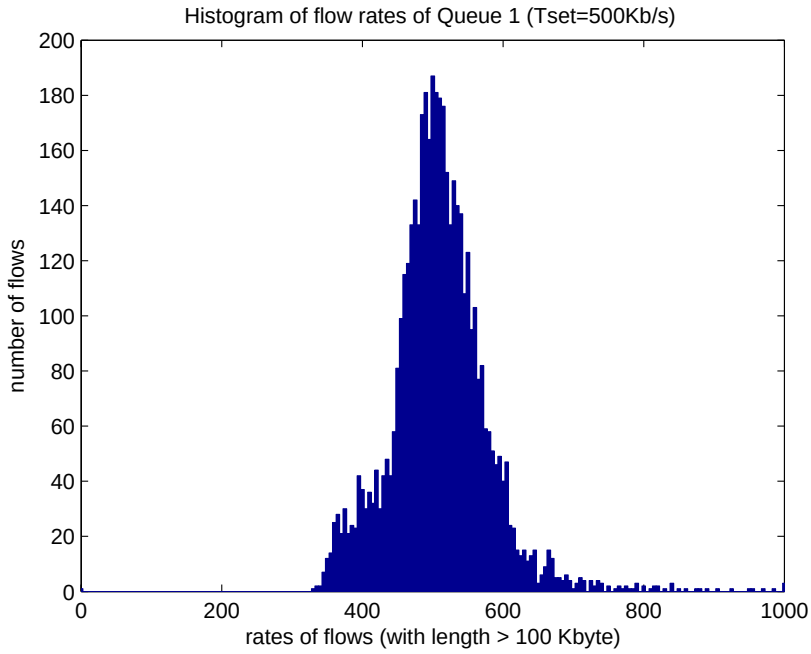

Figure 5.7: Histogram of rate of flows of queue 1 (with length $>$ 100 Kbytes) with $T_{set} = 500Kb/s$ for $\lambda = 112.49$ flows/s in Dynamic DRR simulations in ns.

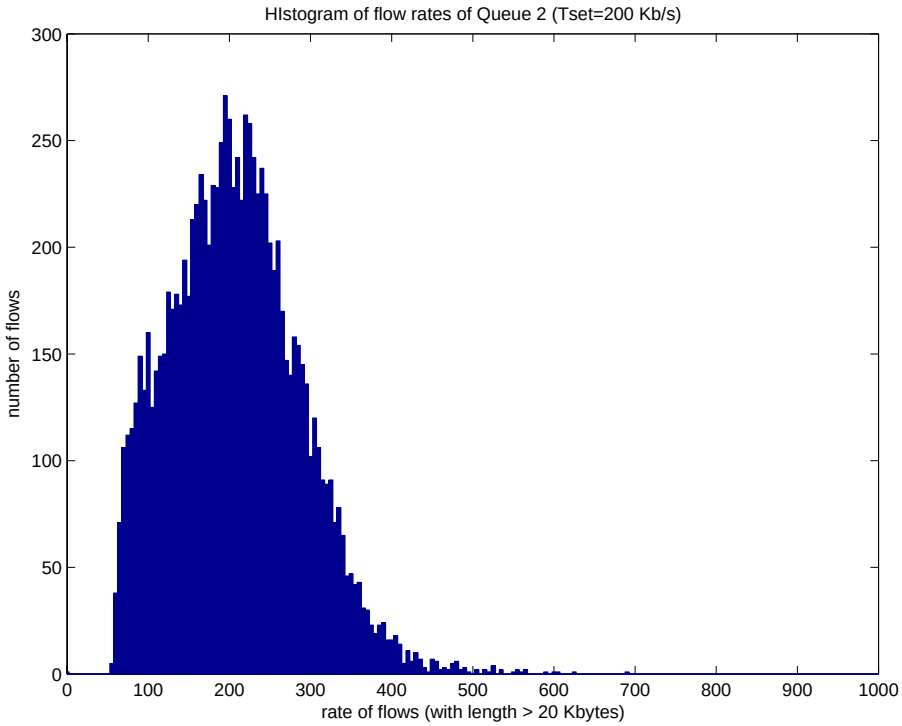

Figure 5.8: Histogram of rate of flows of queue 2 (with length $>$ 20 Kbytes) with $T_{set} = 200Kb/s$ for $\lambda = 112.49$ flows/s in Dynamic DRR simulations in ns.

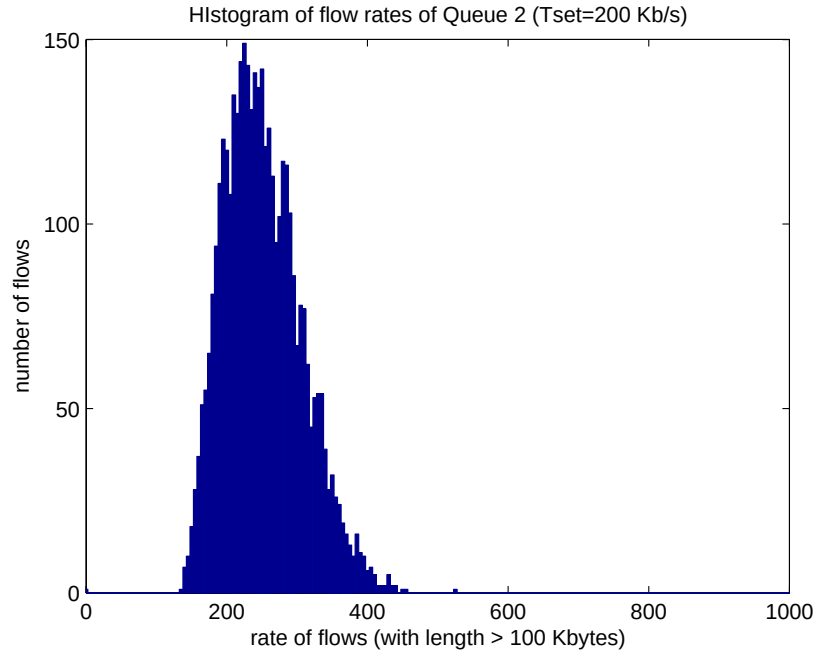

Figure 5.9: Histogram of rate of flows of queue 2 (with length $>$ 100 Kbytes) with $T_{set} = 200Kb/s$ for $\lambda = 112.49$ flows/s in Dynamic DRR simulations in ns.

The second simulation scenario aims to compare the DDRR queue with the FIFO queue, which is mostly implemented in the current network routers. However, when the load of best effort connection increases, the rate received by the other connections decreases due to the behavior of TCP. However, when DDRR queue is used, the flows with different QoS requirements are inserted into different queues, hence, they are not effected by the increase in the load of best effort traffic. In the simulations, the flow rates of class 1 and class 2 traffic are constant but the rate of best effort traffic is increased and the simulations are performed for different loads of best effort traffic. The simulation parameters are tabulated in Table 5.2. The rate statistics of the DDRR queue, when the arrival rate of best effort traffic traffic increases, are given in Table 5.3. Mean rate of FIFO queue when the total load increases is tabulated in 5.4. The rate versus load graph for DDRR queues and for FIFO queue are plotted in Fig. 5.10. The DDRR queue 1 and 2 are not effected by the increase in the load of the best effort queue. However, in FIFO queue, the rates of flows belonging to other classes decrease as the load increases.

Simulation Parameters			
Total Arrival Rate (flows/s)	Arrival Rate of Best Effort Traffic (flows/s)	Arrival Rate of Class 1 or Class 2 (flows/s)	load ($\rho = \lambda \times \beta / C$)
$\lambda_1 = 405$	180	112.49	0.9
$\lambda_2 = 421.872$	196.875	112.49	0.9375
$\lambda_3 = 427.5$	202.5	112.49	0.95
$\lambda_4 = 438.75$	213.75	112.49	0.975

Table 5.2: *Increasing arrival rates of best effort traffic corresponding to 4 different system load values for fixed arrival rates of class 1 and class 2 traffic.*

DDRR Results				
Total Arrival Rate	λ_1	λ_2	λ_3	λ_4
mean(rate) of Queue 1 (T_{set} = 500Kb/s)	509.65	506.89	506.72	509.06
mean(rate) of Queue 2 (T_{set} = 200Kb/s)	199.89	198.77	198.04	198.54
mean(rate) of Queue 3 (Best Effort)	3050.70	94.37	87.06	86.14
std(rate) of Queue 1	83.52	82.91	84.93	86.31
std(rate) of Queue 2	75.9	77.8	77.83	77.32

Table 5.3: *Rate statistics of DDRR Queue when the arrival rate of best effort traffic increases.*

FIFO Queue Results				
Total Arrival Rate	λ_1	λ_2	λ_3	λ_4
Mean(rate) of FIFO Queue	2966.6	141.73	105.96	92.75

Table 5.4: *Mean rates of FIFO Queue when the arrival rate of best effort traffic, thus the total load increases.*

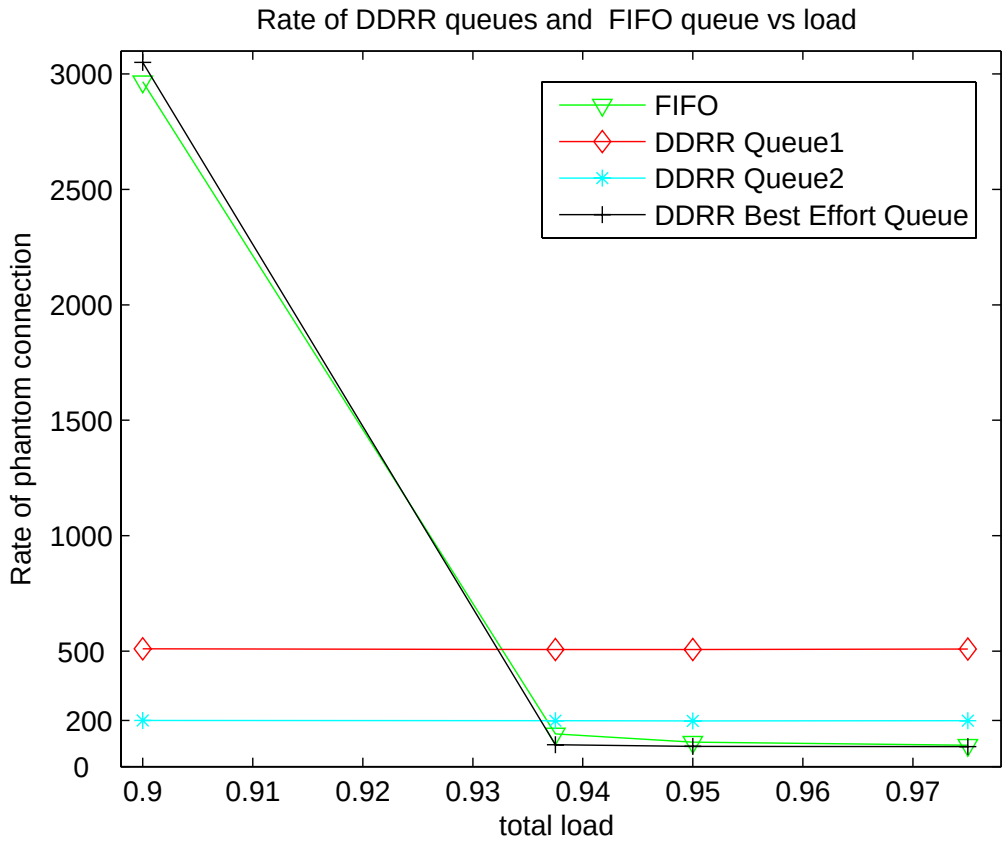

Figure 5.10: Comparison of FIFO and Dynamic DRR: The arrival rate of class 1 and class 2 flows are constant but class 3 (best effort) increases.

In our simulations, we assumed that the propagation delay on the links is almost zero, which is not the case in the realistic scenarios. In real case, each flow in the network will receive a different total delay, which is composed of queueing delay plus the propagation delay, since, each flow traverse different links until the destination. Therefore, the rate of each flow receiving different delays, will be different from the rate of phantom connection.

In our simulations, the QoS criteria is the mean rate obtained in average for each queue. The policy of the controller is determined according to the criteria that is based on minimizing the error between the desired average rate and the current rate of the phantom connection. Therefore, finding the minimum capacity required to meet these requirements is the objective of the learning simulations. However, according to our criteria of setting the rate to desired average, in certain cases, best effort traffic may receive more rate for light load conditions of the upper classes, than the rates received by upper classes. We can deal with these situations by proposing different scenarios for these cases. For example, the rate that the best effort traffic may receive can be limited by a certain higher limit such that best effort traffic rate is always smaller than the rates of upper classes in average. Resultantly, the remaining capacity left by limiting the rate of best effort traffic can be distributed to the upper classes, which will increase the rate of upper classes. However, this scenario, which can be used as an alternative, is different from our scenario, which aims to set the rate to the desired level.

We performed the learning simulations for different arrival rates and for different mean rate values. However, DDRR simulations in ns are performed for fixed arrival rate for each queue, hence a fixed policy is used for each queue. As a future work, the arrival rate of each queue can be detected and policy switching can be performed when the arrival rate of a queue changes.

Chapter 6

Conclusions

For flow level QoS assurances, different classes of elastic traffic are inserted into different queues. These queues are served using Dynamic DRR algorithm whose weights or *Quantum* parameters are adjusted according to the previously obtained policies. These policies use the feedback obtained from the network at each time interval to calculate the capacity required for the next time interval. The policies for different arrival rates and mean throughput requirements are obtained using the M/G/1-PS model of TCP through reinforcement learning simulations and tested using the ns-2 simulator. The rate of each queue are measured by monitoring the throughput of infinite phantom connection at each time interval and average of these measurements are found to be consistent with the average throughput requirements of each traffic class. The capacity left from upper class queues is used by the best effort traffic so that best effort traffic can get more resources when the upper classes are lightly loaded as opposed to static case where the best effort queue uses fewer resources even when there are unused resources.

In the simulations, DDRR scheduler is compared with the FIFO queue as the load of the best effort traffic increases. The first and second class queues of

the DDRR scheduler are not effected from the increase in the load of best effort queue and the average rates remain constant. However, when all classes and best effort traffic are inserted into the FIFO queue, the increase in the arrival rate of the FIFO queue increases the total load causing the average rates of upper classes to decrease.

The policies are obtained for fixed arrival rates therefore, when the arrival rate of a class of flows increase, policy used for the previous policy must be switched to a new policy, which can be simulated as a future work. TCP flows entering a single bottleneck link are modelled using M/G/1-PS model, which is a simple but inaccurate model, since it does not take propagation or processing delay into account. A more accurate model of the system may be used for RL simulations. Moreover, although Gosavi's RL algorithm is simple, certain difficulties are experienced during these simulations such as proper choice of learning parameters. In addition, long run simulations should be executed for the learning algorithm to converge by visiting every possible state a large number of times. Therefore, simpler heuristics may be used to find the controller parameters as a future work.

Bibliography

[1] *Applying Reinforcement Learning to Packet Scheduling in Routers*, Acapulco, Mexico, 12-13 August 2003. In Proceedings of the Fifteenth Innovative Applications of Artificial Intelligence Conference (IAAI-03).

[2] Y. Afek, Y. Mansour, and Z. Ostfeld. Phantom: A simple and effective flow control scheme. *SIGCOMM 26*, 1996.

[3] M. Allman, V. Paxson, and W. Stevens. Rfc 2581: Tcp congestion control. 1999.

[4] Grenville Armitage. *Quality of service in IP networks: foundations for a multi-service Internet*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 2000.

[5] N. Benameur, S.B. Fredj, S. Ousleati-Boulahia, and J. W. Roberts. Quality of service and flow level admission control in the Internet. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 40(1):57–71,.

[6] D. P. Bertsekas. Dynamic programming and optimal control, 2nd ed., Athena scientific. *Telecommunication Systems*, 1 and 2, 2000.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. Rfc 2475 - An architecture for differentiated services. *IETF*, Dec. 1998.

[8] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the Internet. *Technical Report RFC 2309, IETF*, April 1998.

[9] B. Braden, D. Clark, and S. Schenker. Integrated services in the internet architecture: An overview. *Technical Report RFC 1633, IETF*, June 1994.

[10] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol - version 1 functional specification. *Technical Report RFC 2205, IETF*, September 1997.

[11] J. Crowcroft and P. Oechslin. Differentiated end-to-end Internet services using a weighted proportional fair sharing Tcp. *ACM Computer Communication Review*, 28:53 – 69, July 1998.

[12] C. Dovrolis, D. Stiliadis, and P.Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. *in Proc. ACM SIGCOMM*, pages 109–120, 1999.

[13] S. Floyd and V. Jacobson. Random early detection gateway for congestion avoidance. *IEEE/ACM Trans. Networking*, 1(4):397 – 413, August 1993.

[14] S. B. Fredj, D. S.O. Boulahia, and J. W. Roberts. Measurement based admission control for elastic traffic. *Technical Report RFC 2309, IETF*, April 1998.

[15] S. Ben Fredj, T. Bonald, A. Proutiere, G. Regnie, and J. Roberts. Statistical bandwidth sharing: A study of congestion at flow level. *in Proceedings of ACM SIGCOMM, San Diego, CA*, pages 111–122, 2001.

[16] A. Gosavi. Reinforcement learning for long-run average cost. *European Journal of Operational Research*, 155:654–6–74, 2004.

[17] A. Gosavi. A tutorial for reinforcement learning. *http://www.eng.buffalo.edu/ agosavi/tutorial.pdf*, June 2004.

[18] Y. Ito, S. Tasaka, and Y. Ishibashi. Variably weighted round robin queueing for core IP routers. *Proc. IEEE International Performance, Computing, and Communications Conference (IPCCC 2002)*, pages 159 – 166, Apr. 2002.

[19] ITU-T. Recommendation e.800 - Terms and definitions related to quality of service and network performance including dependability. August 1994.

[20] J.Hall and P. Mars. Satisfying qos with a learning based scheduling algorithm. *Proceedings of the Sixth IEEE/IFIP International Workshop on Quality of Service*, pages 171 – 173.

[21] J.Nagle. On packet switches with infinite storage. *RFC 970,IETF*, December 1985.

[22] J.Roberts. Internet traffic, QoS and pricing. *Proceedings of the IEEE*, 92(9):1389 – 1399, Sept.2004.

[23] L. Kleinrock. Queueing systems volume ii: Computer applications. *Wiley-Interscience Publication*, 1976.

[24] L. Kleinrock. *Queuing Systems 2.* Wiley-Interscience, 1976.

[25] Z.G. Li, C. Chen, and Y.C. Soh. Relative differentiated delay service: time varying deficit round robin. *Proc. 5th World Congress on Intelligent Control and Automation*, (6):5608–5612, 15-19 June 2004.

[26] L.Ji, T.N.Arvanitis, and S.I. Woolley. Fair weighted round robin scheduling scheme for diffserv networks. *Electronic Letters,*.

[27] L. Massoulie and J.W. Roberts. Bandwidth sharing and admission control for elastic traffic. *Telecommunication Systems*, pages 185–201, 2000.

[28] P. McKenney. Stochastic fairness queuing. *IEEE INFOCOM*, June 1990.

[29] A. Michalas, P. Fafali, M. Louta, and V. Loumos.

[30] J. Postel. RFC 793: Transmission control protocol. *ACM Computer Communication Review*, Sept. 1981.

[31] J. Postel. Internet protocol. *Technical Report RFC 791, IETF*, September 1981.

[32] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. *In Proc. ACM SIGCOMM*, pages 231–242, 1995.

[33] Network simulator. http:www.isi.edu/nsnam/ns, accessed June 2005.

[34] S.Oueslati and J.Roberts. A new direction for quality of service:flow-aware networking. *Proceedings of NGI 2005,Rome*, June 2005.

[35] J. Sun, G. Chen, K. T. Ko, S. Chan, and M. Zukerman. PD-controller: A new queue management scheme. *GLOBECOM*, 2003.

[36] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *The MIT Press,Cambridge, MA*.

[37] J. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16:185202, 1994.

[38] C. Watkins. Learning from delayed rewards. *Ph.D. dissertation, Kings College, Cambridge, England*, 1989.

[39] J. Wei, Q. Li, and C.Z. Xu. Virtuallength: a new packet scheduling algorithm for proportional delay differentiation. *Proc.12th International Cnferance on Computer Communications and Networks*, pages 331–336, 20-22 Oct.2003.

[40] J. Wroclawski. The use of RSVP with IETF integrated services. *RFC 2210*, 1997.

[41] C. Wu, H. Wu, C. Liu, and W. Lin. A ddrr-based scheduler to achieve proportional delay differentiation in terabit network. *Proc. 19th International Conference on Advanced Information Networking and Applications*, 2:347–350, 28-30 March 2005.

[42] K. Yamakoshi, E. Oki, and N. Yamanaka. Dynamic deficit round-robin scheduler for 5-tb/s switch using wavelength routing. *Proc. 2002 Merging Optical and IP Technologies Workshop on High Performance Switching and Routing*, pages 204 – 208, 26-29 May 2002.

[43] Y.Moret and S.Fdida. A proportional queue control mechanism to provide differentiated services. *Proc.International Symposium on Computer and Information Systems(ISCIS)*, Oct.1998.