TRAFFIC MANAGEMENT IN INTEGRATED SERVICES NETWORKS: SCHEDULING AND RESOURCE PARTITIONING

RAHUL GARG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, DELHI
HAUZ KHAS, NEW DELHI-110016, INDIA

July 1999

TRAFFIC MANAGEMENT IN INTEGRATED SERVICES NETWORKS: SCHEDULING AND RESOURCE PARTITIONING

by

RAHUL GARG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted

in fulfillment of the requirements of the degree of

Doctor of Philosophy

to the

INDIAN INSTITUTE OF TECHNOLOGY, DELHI

HAUZ KHAS, NEW DELHI-110016, INDIA

July 1999

Certificate

This is to certify that the thesis titled **"Traffic Management in Integrated Services Networks: Scheduling and Resource Partitioning"** being submitted by Rahul Garg to the Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, for the award of the degree of Doctor of Philosophy, is a record of bona-fide research work carried out by him under my supervision, and in my opinion, it has reached the standard fulfilling the requirements of the regulations relating to the degree.

The results obtained in this thesis have not been submitted to any other university or institute for the award of a degree or a diploma.

Dr. Huzur Saran
Professor
Department of Computer Science and Engineering
Indian Institute of Technology
New Delhi 110016

To my parents

Acknowledgements

The journey towards my Ph.D. was long and had many diversions. This could not have been carried out without the constant support and encouragement which I received from my parents. I dedicate this thesis to them.

There were many people who helped me in a number of ways in this pursuit and need a special mention.

Firstly, I thank Dr. Srinivasan Keshav who opened my eyes to the intricacies of computer networks. His undergraduate course in networks, which he taught while he was visiting IIT-Delhi from Bell Labs, got me interested in the topic and helped me build a solid foundation. Secondly, I thank Dr. Xiaoqiang Chen whom I met during my summer internship at Bell Labs. Chen gave me support and encouragement in my most difficult times. The work with him forms the most important part of this thesis. I thank my advisor Prof. Huzur Saran for guiding the later parts of my thesis. He provided me support in difficult times and helped me in a number of ways. My work with him forms a significant contribution of this thesis. I thank Prof. Raphael Rom, who was a Sun Microsystems Labs, for guiding my way during the later stages. Prof. Subhashis Banerjee was especially helpful by creating an open and healthy work environment at IIT Delhi. I thank him for taking pains to make sure that students get "unrestricted" access to most of the computing facilities in the CSE Department of IIT Delhi. Finally, I thank Prof. S.N. Maheshwari, whose encouragement right from the undergraduate days prompted me to pursue a Ph.D.

I thank my friend Vishal for providing cheerful company during the dull and depressing days at IIT-Delhi. Vivek who was physically at Santa Cruz, also made his presence felt to me in Delhi, thanks to the newer communication media. I thank all my other friends at IIT-Delhi including Shiv, Sachin, Sharad, Avinash, for their excellent company.

Finally I thank all my newly formed friends at IBM, especially Vinayaka, Johara, Tanveer and Raghavendra who also read the final draft of my thesis and pointed out several "bugs".

Abstract

Traffic characterization, admission control and packet scheduling are important traffic management functions needed for provisioning QoS in Integrated Services Networks. In this thesis, we examine each of these functions in the context of generic networks and virtual (private) networks providing integrated services.

We characterize burstiness of several long traces of video at different time scales using burstiness function. We examine video compressed using JPEG, MPEG and NV's compression algorithms. JPEG video has low short term burstiness and a large long term burstiness, MPEG video has burstiness at short as well as long time scale whereas NV's traffic has only short term burstiness. We discuss the implications of burstiness at different time scale on bandwidth and buffer allocation.

We then focus on deterministic guarantees (which provide a bound on worst case behavior) and compare the performance of leaky bucket traffic model with $X_{min}, X_{avg} I$ traffic model while using optimal admission control tests. We found that the leaky bucket model outperforms the X_{min}, X_{avg}, I model. We show that using good traffic models and optimal admission control tests, even deterministic guarantees provide reasonable network utilization.

An important contribution of this thesis is a new scheduling algorithm called the Recursive Round Robin (RRR) scheduler. The proposed scheduler operates on fixed sized packets and needs simple bit manipulation operations to make scheduling decisions. Thus it can operate at very high speeds. We discuss several variants of the basic scheduler, including the variable size packet scheduler. The delay and fairness properties of the scheduler are analytically derived and discussed in detail. Efficient hardware and software implementations of the scheduler are also suggested. The scheduler has applications in ATM network interface cards, ATM switches and IP routers.

We next show why traditional schedulers are inappropriate for providing bounded delay services in virtual (private) networks. We define the concept of output burstiness and show how generic latency-rate schedulers with bounded output burstiness can provide good end-to-end delay bound in virtual networks. We suggest two variants of the RRR schedulers for virtual networks.

The output burstiness constraint reduces the amount of sharing possible in virtual networks. We propose a capacity resizing approach to improve sharing in virtual networks. The capacity increase requests which are sent at a granularity of session arrivals, are admission controlled, using a new technique called Stochastic Fair Sharing (SFS), which fairly redistributes the free link capacity among different traffic classes. SFS uses the concept of trunk reservation to give high priority to traffic classes with low normalized usage. SFS can be used to achieve better link sharing in virtual networks. We present simulation results for a single link and a twelve node network. SFS achieves fair sharing on a single link and max-min fair sharing in a network with fairness index between 0.96 and 0.99, as compared to an index between 0.66 and 0.97 for simple schemes like FCFS. The bandwidth penalty for using SFS was less than 5% and the signaling load for SFS in a network was less than 100 messages per second per router.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditional means of communications employ different networks for different services. For instance, there is a separate telecommunications network for providing voice telephony services. Similarly, there are different networks for television, radio and data transport services. These networks were independently developed and are engineered for the specific service they provide.

With the recent rapid advances in computer and communications technology, different communications services are converging into some form of *Integrated Services* communication. Today, computers can generate and process information in the form of different media like voice, video, text and data and use the same communication networks to transport this information. In such a scenario, different applications can potentially provide different services using the same underlying Integrated Services Networks (ISN).

In addition to emulation of traditional services like voice telephony, video broadcast and data transfer, future Integrated Services Networks are expected to support several novel applications like video conferencing, distributed games, real-time monitoring and control of remote systems etc. Most of these applications need to synchronize their action in real-time, and for this they require that each of their sub-tasks complete in a timely fashion. For instance, a video display application displaying 30 frames per second require that the display of a frame does not take more than 33ms in the worst case. Similarly these applications need a performance bound (also called

Quality of Service requirement) from the underlying communication network. Such a performance bound is critical for correct functioning of the applications.

Consider a computer telephony application. Voice digitized at one end is packetized and sent to the other end via a packet network. At the receiving end, the application receives the voice packets and reconstructs the original voice. For such an application to work correctly, additional constraints on network performance are needed. If voice is digitized (without any compression) at telephone quality, its transmission would require a dedicated bandwidth of 64 Kbps. Even if the network makes the required bandwidth available to the application, the end-to-end delay of successive packets across the network may be highly variable because of variation in queue lengths at intermediate nodes in the network. If the packet delay is not bounded, then the application would not know how long to wait for a packet before reconstructing the voice. If the network guarantees a bound on the maximum delay incurred by the packets, the application can buffer the packets appropriately before regenerating the voice. However, if the bound on the delay is large, the humans talking using this application would perceive a large delay which may not be acceptable to them. In general a small bound on maximum delay is desirable. Network may drop packets because of various reasons, and it is also desirable to have a bounded packet loss rate.

Providing guaranteed QoS is vital to support various real-time services in Integrated Services Network. The telephony application will not be able to send voice if the desired bandwidth is not available. Distributed real-time games will become highly unpredictable if there is no bound on packet delays. They may even become unfair to players having large packet delays. A lost packet or a delayed packet can play havoc on a real-time control system.

The QoS parameters defined for a connection (also called flow or session in a connection-less network) between two (or more in case of multicast) communicating applications are [73]:

Bandwidth. It identifies the amount of bandwidth which should be exclusively reserved for the connection. Applications like video conferencing or voice telephony require that a minimum bandwidth be available to them even during peak congestion periods. If this bandwidth is not available, the applications will simply fail to work.

Delay. Packets sent by applications incur a delay before they are received at the other end. This delay has a fixed component and a variable component. The fixed component is because of the signal propagation delay and constant processing delay at intermediate switches and routers. The variable part of the delay is due to the packet queues at various contention points in the network [5]. Most of the applications require the delay to be as small as possible. The *delay* QoS parameter, bounds this packet delay for a connection. Depending upon the definition of QoS parameters, the delay parameter could bound the maximum packet delay of a connection, or the mean packet delay, or a percentile of the delay. A low value of delay results in a faster response time between communicating applications, and results in better interactive performance.

Delay-jitter. It is defined to be the difference between the maximum and the minimum packet delay of a connection. For some applications such as video on demand, the delay jitter is more important than the absolute value of the maximum delay. The delay jitter bounds the maximum amount of (playback) buffering needed in these applications.

Loss-rate. Some of the packets sent on a connection may become corrupted because of transmission errors and need to be dropped. Some other packets may also have to be dropped because of buffer overflows caused by transient congestion in the network. The QoS parameter loss rate bounds the loss rate of packets because of these problems.

QoS guarantees can be provided using enhanced traffic management functions. Traffic management is concerned with the problem of managing heterogenous traffic in Integrated Service Networks such that a diverse range of service requests are satisfied as efficiently as possible [73]. In traditional data networks, most of the service requests are homogeneous and congestion is the most severe impediment to good performance. Therefore, most of the traditional research on traffic management has focused on the problem of congestion control. However, in the context of Integrated Services Networks, traffic management has assumed a new significance, as the issues related to QoS need to be solved. In this thesis we examine the traffic management issues for efficient provisioning of QoS in Integrated Services Networks. The key components to provide QoS are traffic characterization, admission control and scheduling.

Depending upon the type of guarantees provided on these parameters, the QoS guarantees may be classified into four classes: class based approaches, deterministic QoS guarantees, statistical QoS guarantees and ad-hoc assurances.

Class-based Approaches: In these approaches, an application can request for QoS only from a predefined set of QoS classes. This set should be generic enough to fulfill the requirements of each potential application. The set contains classes like MPEG video, audio, telnet traffic, world wide web traffic etc.

Deterministic Guarantees: In deterministic QoS guarantee, mathematically provable worst case bounds on QoS parameters like delay, jitter and loss rate are provided. The QoS guarantees are quantitative. Moreover, whether the network is honoring the QoS commitments or not, can easily be checked by monitoring the traffic and its delay and loss pattern. Deterministic guarantees bound the worst case behavior and are therefore expected to reserve resources for the extreme case which may be very unlikely. For instance, they may reserve bandwidth for a connection at its peak rate. This may result in lower network utilization. This was a major criticism of deterministic guarantees.

Statistical Guarantees: Instead of bounding worst case parameters, statistical QoS guarantees provide a bound on percentiles. In many cases, it is sufficient to ensure that the expected fraction of packets exceeding the delay bound is very small (say 10^{-7}). These guarantees are based on a statistical model of the traffic. Statistical guarantees may result in better network utilization as compared to deterministic guarantees. However, because of their statistical nature, it is difficult to measure the validity of statistical guarantees. It is also difficult to police traffic based on a statistical model.

Ad hoc Guarantees: With the intent of minimizing implementation complexity in a large datagram network, a differentiated services architecture has been proposed in [22, 95, 94]. This architecture achieves scalability by aggregating traffic classification state which is conveyed by means of packet marking using a special field in the packet header. Packets are classified and marked to receive a particular per-hop forwarding behavior on nodes along their path. Sophisticated classification, marking, policing, and shaping operations are only implemented at network boundaries or hosts. Network resources are allocated to traffic streams by service provisioning policies which govern how traffic is marked and conditioned upon entry to such a network, and how that traffic is forwarded within that network. The QoS is provided to aggregate traffic in a particular class. There is a service level agreement between a customer and a service provider which specifies the forwarding service a customer should receive. Similarly there is a traffic conditioning agreement which specifies the amount of traffic (of each class) which a customer is expected to generate. The lifetime of these agreements are long (say a month or even more). The service provider provisions its network to make sure that the service level agreements of its customers are honored. This architecture achieves scalability by pushing as much complexity to the network edges as possible. Admission control is also expected to run only on aggregate traffic and on a longer time scale.

In this thesis we focus on deterministic QoS guarantees. Some of our proposed algorithms and techniques are also applicable in the context of class-based approaches and ad-hoc guarantees, but their analysis and simulation is mostly done in the context of mathematically provable deterministic QoS guarantees. The deterministic service model has been adopted in the ATM forum [1] for CBR and VBR service and in the integrated services [20, 127, 108, 110] working group of IETF for the guaranteed service class [108].

For guaranteeing QoS, there must be a way by which the network can limit its traffic in case of overload. By controlling the entry of the traffic into the network, the network can protect itself from overload and provide required performance guarantee to its traffic. This control is done in traditional networks using different closed loop flow control algorithms. These flow control algorithms operate on a time scale of several round-trip times. Therefore, reacting to overload may take several round-trip times. The delay bounds required by the real-time applications are typically in the range of a round-trip time or less. Therefore, the closed loop techniques are not suitable for such traffic. An open-loop control method is used to control the traffic of such applications. The application sending traffic characterizes its traffic to estimate the resources (such as bandwidth, buffer) needed in the network. The application then sends a request to reserve the required resources [132, 2, 1, 19] in the network for its traffic. The application begins its data transfer only if the required resources have been reserved. In case the required resources are not available in the network, the application gets a *busy tone*, signalling this unavailability. The application tries to reserve resources at a later time, when the network load is low. This is very similar to getting a busy tone in a telephone network. The traffic is admitted into the network only after the reservation of its resources. This process is called admission control.

In addition, there is a policing function in the network [73]. The policer checks if the applications are sending data at a rate which is in agreement with its traffic characterization. If an application sends data at a higher rate, its packets are either discarded or marked as non-compliant by the policer. The non-compliant packets are discarded as soon as there is overload or congestion in the network. Thus, by using admission control and policing, the amount of traffic entering the network is controlled in an open loop fashion.

Finally, there are packet scheduling algorithms at the switch which decide the order in which packets are sent on a link. In a packet switching network, packets from different connections interact with each other at each switch. Without proper control, these interactions may adversely affect the network performance experienced by clients. The scheduling algorithm at the switching nodes, which controls the relative ordering in which packets from different connections are serviced, determines how packets from different connections interact with each other.

Together with traffic modeling and connection admission control algorithms, schedulers are the most important components for providing QoS guarantees. The connection admission control algorithms reserve resources during connection establishment time. The packet scheduling algorithms allocate resources according to the reservation during data transfer. Three types of resources are being allocated by a scheduling algorithm: bandwidth, promptness and buffer space, which in turn affects three QoS parameters: throughput, delay and loss rate.

In the first half of this thesis, we examine these three important aspects of traffic management in Integrated Services Networks: Traffic characterization, admission control, and scheduling algorithms. In the second half we focus on traffic management for Integrated Services Virtual Networks.

Virtual Networking is another important step in the evolution of communication networks. A virtual network is a network overlaid on a physical network of a different topology. The physical network may be based on legacy systems which cannot be upgraded, and may only provide a small set of services as compared to the virtual networks laid over it. Virtual networking allow quick deployment of new services over legacy networks, ease network operation and management by hiding the unnecessary details and presenting a simplified topology, allow development of experimental protocols in a controlled and safe environment and ease inter-operability between networks of different types.

A virtual network consists of virtual links which are realized either by a direct physical link, or by a logical path in the physical network. These virtual links join the virtual nodes of virtual networks. Some examples of virtual networks are an ATM network carrying virtual connections over virtual paths [16] (for simpler network operation and management), IP over ATM networks (for inter-operability), virtual private networks (VPN) in the Internet [109] and IPV6, and the MBONE and 6-bone virtual networks [30, 82] over the Internet (for deploying multicast services over the legacy network).

In an ATM network, the virtual paths are often used to realize any virtual network using a given physical network. A virtual path between two nodes acts logically like a direct link between them. All the traffic entering a virtual path at its entry node gets routed to its exit node as if there was a direct link connecting the two nodes. The use of virtual paths helps reduce the complexity of the network by aggregating several virtual connections into a single virtual path. This makes the network more scalable as the backbone switches only need to maintain the state corresponding to virtual paths passing through them. The backbones do not need to maintain state corresponding to individual connections. Use of virtual paths also simplifies operation and management of the network.

The design of routing protocols (and many other protocols) in telephone networks is often based on the assumption that all the central offices in an area are connected to each other forming a complete graph. As the areas become bigger, it may not be possible to lay a physical trunk between all the central office pairs. In such situations, a completely connected virtual network is overlaid on the physical network.

In an enterprise wide private network, several offices of an enterprise are traditionally connected using leased lines. Now the leased lines are beginning to get replaced by public networks to achieve the same functionality. Instead of a leased line connecting two offices of an enterprise, virtual private network routers are placed in the enterprise. These routers are connected to a common public network such as the Internet. VPN routers tunnel their traffic through the public network giving the impression that they are connected by a leased line.

Similarly IP over ATM is another example of a virtual IP network overlaid on an ATM network.

MBONE makes extensive use of tunnels to overlay a multicast capable virtual network on an IP network which does not have multicast support built in it.

The research related to QoS in virtual networks has been limited. In the second part of this thesis we examine the issues pertaining to Integrated Services over virtual networks.

1.1 Our Research and Related Work

Digital video is an important component for multimedia applications. Video traffic is bursty in nature. The variation of its bit-rate with time is large. Therefore, efficient transport of real-time video in an Integrated Services Network requires a good characterization of its bit-rate process. Chapter 2 of this thesis is devoted to video traffic characterization.

The bit-rate of VBR video has been modeled in [88] as a first order autoregressive process with marginal Gaussian probability distribution function and an exponential autocorrelation function. In [56], a more sophisticated autoregressive moving average process (ARMA) was used to model the video traffic. Similarly a number of other models, including the self-similar and long range dependent model have been proposed in [113, 58, 81, 90, 45, 98]. Although it is clear how these models may be used to construct synthetic video traces for simulations, or to estimate effective bandwidth [29] for statistical QoS guarantees, it remains to be seen how these models may be used in the context of deterministic QoS guarantees.

The ATM Forum uses the leaky bucket traffic model [1] for its deterministic QoS guarantees and the Tenet group has proposed X_{min}, X_{avg}, I model [34] for deterministic guarantees. We use these models to characterize long traces of video. We have examined video traffic generated by three different coding algorithms namely MPEG, JPEG and compression algorithm of software NV. We have considered one to two hour long traces of five video sequences of different types, ranging from a lecture in a classroom to a basketball match. We also characterize the burstiness of video traffic at different time scale using the burstiness function. We develop qualitative insights to choose appropriate leaky bucket traffic descriptors for video traffic.

A major criticism of the deterministic guarantees is because of the fact that they require the network to reserve resources for the worst possible case. This results in low network utilization. We show in Chapter 3 that with the choice of proper traffic model and optimal admission control tests, high network utilizations can be achieved. The earlier admission control tests for deterministic guarantees were suboptimal. We suggest new optimal admission control tests for the EDF scheduling algorithm. Using these tests and video traffic traces we analyze the performance of deterministic QoS guarantees. We show that the video traffic alone could result in high utilization of the network.

While the admission control and resource reservation protocols reserve the required resources for traffic streams, the packet scheduling algorithms on the switches and routers actually allocate them. These algorithms allocate the bandwidth, promptness and buffer resources at switches and routers which control the three QoS parameters: throughput, delay and loss rate.

The results related to scheduling algorithms in the context of hard real-time systems and queueing systems are not directly applicable to packet scheduling algorithms in Integrated Services Networks. A typical real-time system is modeled as a single server with periodic jobs where the job delay is bounded by its period [114]. However in a network, traffic is bursty, packets typically travel a number of hops before reaching their destination, and traffic dynamics is far more complex than a single server environment. Queueing analysis is often intractable for realistic traffic models. The classical queueing analysis usually studies average performance for aggregate traffic [74, 75, 126], while in an integrated services network, performance bounds need to be derived on a per connection basis [31, 76].

A packet scheduler in an Integrated Services Network should have the following main properties:

Bandwidth Guarantee: The scheduler should provide a minimum bandwidth guarantee to individual traffic streams.

Delay/Jitter Bound: For deterministic QoS guarantees, the scheduler should provide a bound on the maximum queueing delay (and jitter) incurred by packets of given traffic streams. It should be possible to compose these "per-node" bounds to get tight end-to-end bounds on packet delays.

Buffer Bound: To guarantee a zero loss rate to traffic streams, the scheduler should provide an upper bound on buffers needed at nodes. In most of the cases this bound follows directly from the traffic descriptor and bandwidth and delay guarantees.

Efficiency: In order to provide a given bound on delay, the scheduler should not over-allocate bandwidth to streams. Efficient schedulers allow more traffic streams to be admitted and increase the network utilization.

Fairness: The unused link bandwidth should be distributed to active traffic streams in proportion to their bandwidth guarantees.

Implementation Complexity: The time available for making a scheduling decision is of the order of a few hundred nanoseconds. It should be possible to implement the scheduling algorithm in a switch or a router operating at such speeds.

A number of packet scheduling algorithms for Integrated Services Networks have already been proposed in the literature [47, 130, 128, 120, 26, 86, 100, 131, 8, 9, 129, 52, 115, 116, 119, 122, 68, 51, 99]. Some of these algorithms were first designed for scheduling packets of variable sizes and were complex [26, 86, 100, 8]. In order to make a scheduling decision, these algorithms may need to perform $O(N)$ operations in the worst case, where N is the number of traffic streams being scheduled. Minor modifications in these scheduling algorithms result in faster schedulers [52, 129] with similar QoS properties. These schedulers require $O(\log(N))$ operations to schedule a packet. These algorithms typically tag each packet with a virtual timestamp and then select the packet with the smallest tag. Tag computation may require operations like multiplication and division and selecting a packet requires sorting via a priority queue. Implementing these algorithms at high speeds poses a difficult challenge. Among other schedulers amenable to high speed implementation, some have poor delay and fairness properties [87], some are not scalable for fine rate granularity [99] and some may need to over-allocate rate resulting in poor utilization of link bandwidth [68].

In Chapter 4 of this thesis, we propose and describe a new scheduling algorithm called *recursive round robin scheduler* (RRR), which is optimized for scheduling fixed size packets, like cells in an ATM network. The scheduler has the required QoS properties for deterministic guarantees. For a compliant stream of ATM cells of rate r, and bucket size σ, the scheduler provides a delay and jitter bound of $\frac{1}{r}(\sigma + c)$, where c is a small constant. The work-conserving version of RRR satisfies the fairness property. We analytically derive the bounds for the fairness index. The scheduler can also be used for hierarchical scheduling in a link sharing [38] environment. RRR only requires simple bit manipulation operations to make a scheduling decision. Therefore, very high speed implementations of RRR are possible. We outline a simple software implementation of RRR for network interface cards and a hardware implementation for high speed ATM switches. We generalize the basic RRR algorithm for scheduling variable sized packets. We also list the delay and fairness properties of the variable sized packet scheduler. Therefore, the RRR schedulers may be used in ATM network interface cards, ATM switches, IP routers and IP host adapters.

In the second part of this thesis, we discuss traffic management in the context of Integrated Services Virtual Networks.

Since virtual networking is a relatively newer concept, the research related to QoS in virtual networks has been limited. The early work on virtual networks has focused on developing new services or managing the virtual networks more efficiently. Virtual Private Networks (VPNs) [39, 50, 66] are increasingly being used by organizations. However, the research related to VPNs has been focused primarily on security enhancements [109], and operation and management. The Geoplex system [91] is an IP based service platform that offers support for rapid automated deployment and management of overlay networks. Similarly the Genesis project [124] and the Netscript project [36] aim towards developing programmable virtual networks. The need for enabling QoS in virtual networks is increasingly being realized.

Packet scheduling algorithms are the key to providing QoS in any network. The research related to scheduling algorithms for virtual networks has been limited. In [38] the concept of link sharing has been proposed and scheduling algorithms to implement link sharing have been proposed in [9, 8, 129]. These algorithms hierarchically partition the link bandwidth into classes and provide bandwidth and delay guarantees to traffic streams from each subclass. However, it is not clear how these scheduling algorithms can be used for integrated services in virtual networks.

The virtual network traffic is aggregated in the virtual links (or tunnels). Ideally, the physical network should only keep the state information corresponding to the aggregate traffic of virtual links and treat the virtual link traffic as if it was a single session in the physical network. In the physical network, it is undesirable to maintain state information corresponding to each session of virtual network.

Under these assumptions we show in Chapter 5 that traditional work-conserving schedulers cannot provide bounded delay service in a virtual network. The problem arises because traffic of a number of sessions sharing a virtual link in a virtual network is aggregated and tunneled through the physical network, which cannot isolate the traffic of well behaved sessions from that of misbehaving sessions. We define a term called output burstiness and show that this problem can be solved by regulating output burstiness on virtual links. Using the theory of latency rate servers [117], we show that latency rate servers with bounded output burstiness, may be used in a virtual network to provide bounded delay service. This gives a method to design a generic class of scheduling algorithms for virtual networks. We discuss how variants of the RRR scheduling algorithm may be used in virtual network.

The output burstiness constraint limits the rate at which traffic may be sent on virtual links of a virtual network. As a result, some packets may have to wait in queues even while the physical link is idle. This reduces the overall throughput of the network. In Chapter 6 we suggest a capacity resizing approach to improve sharing among virtual networks.

In this approach the capacity of virtual links may be adjusted dynamically by sending *increase* or *decrease* requests in the network. These capacity resizing requests are sent at a time scale of session arrivals. The increase request is admission controlled in the physical network. Carrying out this admission control naively (such as on FCFS basis) may result in unfair capacity allocation on virtual links. Fairness is an important issue especially in the context of virtual private networks (VPN) where customers pay for a given capacity on virtual links.

We propose a new scheme called Stochastic Fair Sharing (SFS) to carry out fair link sharing and fair sharing among leased virtual links. In the link sharing environment, SFS decides which sessions to accept and which to reject depending upon the current utilizations and provisioned capacities of the classes. After accepting a session, the underlying scheduler weights are adjusted to reflect the change in capacities. SFS gives protection to classes with low session arrival rate against classes with high session arrival rates by ensuring them a low blocking probability.

In case of multi-hop virtual links, if the capacity increase requests are admission controlled using the SFS admission control algorithm, the capacity allocations converge to the max-min fair allocations. We describe the SFS scheme in detail in Chapter 6. We also present simulation results for a single link as well as a network using different traffic mix.

The critical assumptions in the design of SFS are:

- Individual session bandwidth requirement is small as compared to virtual link capacity.
- Most of the sessions have small holding time.
- The session arrival process is not very bursty.

The scheme is simple, efficient, robust and results in fair sharing of link capacity when used for link sharing, and results in max-min fair sharing of capacities when used in a network. The potential applications of SFS are fair and efficient resource sharing in telecommunication networks, ATM networks, virtual private networks (VPN) and integrated services or differentiated services based IP networks.

1.2 Summary of Contributions

The contributions in this thesis broadly consist of:

- **Traffic Characterization and Admission Control**
  - Characterization of long traces of video compressed using JPEG, MPEG coding algorithms and NV software, using leaky bucket traffic model, X_{min}, X_{avg}, I traffic model and burstiness function.
  - Performance evaluation of deterministic QoS guarantees and improved admission control tests.
- **Scheduling**
  - A new scheduling algorithm called the recursive round robin scheduler (RRR) for fixed size cells.
  - Detailed analysis of QoS related properties of RRR and adaptation of RRR for variable sized packets.

- **Scheduling in Virtual Networks**
  - Identification of problems with traditional schedulers when used in virtual networks.
  - Definition of the term called output burstiness and using it to design a class of schedulers (including two variants of RRR) for virtual networks.
- **Fair sharing in virtual networks**
  - A new resizing approach called Stochastic Fair Sharing (SFS) along with admission control algorithm for fair sharing in virtual networks.
  - Detailed simulations of SFS for a single link and a large network scenario.

This thesis is organized as follows. In Chapter 2 we characterize traces of video compressed using different coding algorithms. We develop qualitative methods to characterize video traffic using the leaky bucket traffic descriptor. In Chapter 3 a brief survey of service models for integrated services network is given. We then focus on deterministic guarantees and develop optimal admission control tests for the EDF scheduling algorithm. Using the optimal admission control tests and long traces of digital video, we show that the leaky bucket traffic model is better than the X_{min}, X_{avg}, I traffic model. We finally evaluate the performance of deterministic guarantees using optimal admission control algorithms and long traces of digital video. In Chapter 4 we present the complete design and analysis of the RRR scheduling algorithm. In Chapter 5 we discuss the difficulties encountered while using the traditional schedulers in a virtual network. We suggest a method to design a class of schedulers for virtual networks using the concept of output burstiness. Finally in Chapter 6 we present the details of proposed Stochastic Fair Sharing (SFS) technique to carry out fair sharing among virtual networks. The thesis concludes in Chapter 7.

Chapter 2

Video Traffic Characterization

It is important to understand the nature of video traffic in order to design and engineer integrated service networks which efficiently support real-time video transport. Parameters like average rate, ratio of peak rate to mean rate and burstiness of video have significant influence on network design. For instance, if ratio of peak rate to mean rate of video traffic is very high, and video is significantly bursty over a long time scale, then very high traffic utilization with to video traffic alone may not be achievable. In this case, it is desirable to design the network such that non real-time traffic, or less bursty real-time traffic can make use of the residual capacity of the network left from the video sources.

In this chapter, we study the characteristics of video data compressed using standard coding algorithms, namely JPEG, MPEG, and also that used by the video conferencing software NV. We analyze a wide range of video sources, from movies to a class lecture. Most of the traces are longer than one hour. We characterize the bit rate of the traces using the leaky bucket model. We also show a method for choosing appropriate leaky bucket parameters. The burstiness function is used to characterize the burstiness of the video traffic at different time scales. Our studies indicate that JPEG compressed video has very little short-term burstiness. MPEG and NV video traffic shows high burstiness over small time scales. JPEG and MPEG video exhibit some burstiness over long time scales, whereas NV has no burstiness over long time scales. It is found that, for constant quality JPEG and MPEG compressed video, the leaky bucket parameters depend upon the contents of the video. For JPEG video, the service rate is mainly determined by the peak rate, and for MPEG the service rate is given by the peak rate of the smoothed MPEG stream. Although the traffic generated by NV can be characterized independently of the actual video, one needs information about QoS to come up with a good traffic descriptor. The target sending rate of the software NV determines the service rate for its traffic if the required delay bound is high.

2.1 Introduction

It is well known that constant quality video has variable bit rate and is bursty. The burstiness of the traffic is related to the statistical multiplexing gain. The presence of a larger variety of compression algorithms makes the task of traffic characterization difficult. Since the video traffic streams have a complex structure, their effect on the network is much more complex than that found by simple, analytically tractable traffic models.

A first simplistic model of VBR video traffic appears in [88] where the video traffic is modeled as a first order autoregressive process with marginal Gaussian probability density function and an exponential autocorrelation function. A more sophisticated autoregressive moving average process (ARMA) was used to model the video traffic in [56]. Similarly, there are number of other traffic models for VBR video that have been proposed [113, 58, 81, 90, 45, 98]. Although it is possible to construct synthetic traces of video using these models, which may be used for various simulations, it is not very clear how such characterization can help in network management functions like admission control and policing. Also most of these studies have characterized only one particular coding algorithm.

In this chapter, we characterize the video traffic using two widely known traffic models - Leaky Bucket [43] and Tenet group's X_{min}, X_{avg}, I model [34, 33]. We study bandwidth and buffer assignment for leaky bucket model in great detail. Leaky bucket is likely to be an appropriate model to characterize the traffic. We give a simple and effective method of choosing leaky bucket parameters for video traffic. We also characterize the burstiness of the traffic at various time scales using the burstiness function as defined by Cruz [24].

The traces used for this chapter are from a variety of sequences and encoding algorithms. The sequences were chosen to represent a range of sequences present in real life (broadcast TV, class lecture, full length movie). We study standard compression algorithm like JPEG, MPEG, and traces of NV.

JPEG is a compression standard which was originally designed by the Joint Photographic Experts Group for coding and storing still photographic images [125, 104]. Since then it has found applications in many other fields, including digital video. To encode digital video, each frame is compressed independently as a still picture using the JPEG compression algorithm. The video consists of a series of independently compressed JPEG pictures, which makes the algorithm an intra-frame compression algorithm.

MPEG is a compression standard designed for storage and transmission of video [42, 62, 63]. It is an inter-frame coding algorithm that exploits the spatial and temporal redundancy of the video to achieve the compression. It also uses a DCT-based coding as the JPEG algorithm. The frames are classified into 3 types : I-frames, P-frames, and B-frames. I-frames are coded as still pictures and are independent of other frames. This makes I-frames the largest in size. P-frames carry only the changes between the last I-frame or P-frame and the current frame. B frames, which are the smallest in size, carry changes in the current frame with respect to recent or future I-frames or P-frames. A typical order of frames is I-B-B-P-B-B-P-B-B-I-B-B-...

NV is a popular software video tool used over the Internet to hold video conferences at low bit rates [41]. The NV software tries to limit the bit rate while sending video. This bound can be selected by the user, and is generally set to 128 Kbits per second for sending data over the Internet. To achieve this rate control, the NV software measures the actual rate at which it is sending the data. When it realizes that it has sent too much data, it becomes inactive for about 1-2 seconds. This reduces its mean sending rate. The coding algorithm used by NV is a variant of Harr transform, which uses conditional replenishment for individual blocks.

In this chapter we show that JPEG compressed video has very little short-term burstiness. MPEG and NV video show high burstiness over small time scales. JPEG and MPEG video streams exhibit burstiness over long time scales, whereas NV shows no burstiness over long time scales.

We also show that for constant quality JPEG and MPEG compressed video, the leaky bucket parameters depend upon the contents of the video. For JPEG video, the service rate is mainly determined by the peak rate, and for MPEG the service rate is given by the peak rate of the smoothed MPEG stream. The traffic generated by NV can be characterized independently of the actual video. The service rate for NV depends upon the QoS needed by the application. The target sending rate of NV determines the service rate for its traffic if the delay bound is large.

The chapter is organized as follows. Section 2.2 discusses the leaky bucket model, the X_{min}, X_{avg}, I traffic model and the burstiness function, which is used in characterizing the traffic. Section 2.3 discusses the sources of the traces, and the contents of the actual video streams used. Sections 2.4, 2.5 and 2.6 discuss the characteristics of the traffic generated by JPEG, MPEG and NV coding algorithms. Section 2.7 quantitatively analyzes the suggested approach to choose appropriate leaky bucket traffic descriptor. Discussion of the results is presented in Section 2.8. The chapter concludes with Section 2.9.

2.2 Traffic Models

The leaky bucket mechanism has been widely used to characterize and police the traffic [105, 43, 28, 13]. It has two parameters - the bucket size σ and the service rate ρ. The bucket size specifies the maximum amount of data which may be sent by the traffic source as a burst. The service rate specifies the maximum long term average rate at which the source may send its data. A traffic stream is said to comply with a leaky bucket descriptor having parameters (σ, ρ) when the amount of data carried by the stream in any interval of length I is bounded by $\sigma + \rho I$. In other words, the descriptor bounds the bit rate so that no burst can carry more than σ bytes of data, and the long-term average rate is bounded by ρ. The peak rate using the leaky bucket descriptor remains unspecified since a burst of size σ may be sent by the source at any rate. Thus the peak rate of traffic may be equal to the link speed on which it is being sent.

Two leaky bucket descriptors are used to specify the peak rate along with the average rate. Thus there are two tuples $(\sigma_1, \rho_1), (\sigma_2, \rho_2)$ characterizing the traffic, with $\sigma_1 < \sigma_2$ and $\rho_1 > \rho_2$. The parameter ρ_1 specifies the peak rate and ρ_2 specifies the average rate. The value of σ_1 is usually small to accommodate small bursts of data (usually 4-5 packets or cells) sent at peak rate. The burst size is specified by σ_2 as in the single leaky bucket descriptor. A traffic stream compliant to a two leaky bucket descriptor should be individually be compliant to both the leaky bucket descriptors.

Any traffic can be made compliant with another leaky bucket descriptor with the same rate and smaller bucket size by suitably buffering and delaying some of its packets. If a traffic complies a leaky bucket descriptor with parameters (σ_1, ρ), then it can be made compliant with parameters (σ_2, ρ) $(\sigma_2 \leq \sigma_1)$ by suitable regulation. The maximum buffer space needed to do this would be $\sigma_1 - \sigma_2$ and the maximum delay incurred by any packet due to this reshaping of traffic would be $\frac{\sigma_1 - \sigma_2}{\rho}$. In the extreme case when the output of the regulator is a smooth traffic having very small bucket size, the maximum smoothing delays incurred by the traffic would be $\frac{\sigma_1}{\rho}$, and the buffer space required to do this smoothing would be σ_1.

In the Tenet scheme [34, 33], the traffic is described using four parameters, $X_{min}, X_{avg}, I, S_{max}$. X_{min} is the minimum inter-packet spacing of traffic generated by the source. X_{avg} is the average inter-packet spacing of traffic generated by the source. I is the interval over which X_{avg} is measured. The minimum value of X_{avg} is taken for all intervals of length I. This gives the worst case average rate over all intervals of length I. S_{max} is the maximum packet size sent by the source.

Given the above traffic descriptor, the peak rate of the traffic is $\frac{S_{max}}{X_{min}}$. The average rate is $\frac{S_{max}}{X_{avg}}$. This traffic descriptor is not appropriate because according to this descriptor the peak rate and average rate depend upon the maximum packet size and the minimum (or average) inter-packet spacing. If the traffic source generates one packet of very large size and other small packets with small inter-packet spacing, the peak rate and the average rate computed will be much more than the actual peak and average rate.

In order to have a reasonable representation of the traffic, in rest of the chapters we assume that S_{max} is not very large as compared to other packets generated by a traffic source. The traces of traffic which we use later for analysis have this property.

Cruz in [24] has presented a generalized model to characterize the burstiness of the traffic by bounding its worst case behavior. The framework developed by him can be used to provide local deterministic guarantees for a variety of service disciplines. The burstiness function of a traffic source is defined as :

$$b(t) = \text{Maximum amount of traffic sent in any interval of length } t.$$

The burstiness function for a given traffic trace is unique. The value of burstiness function at any point t can be obtained by sliding the trace through a time window of size t and finding the maximum amount of traffic sent in any such window. Since burstiness function measures the worst case average rate over all time scales, it is the most general bounding traffic model which can be used to provide deterministic performance guarantees. However, it would be impractical for a traffic source to completely specify its burstiness function. As a result simple models with few parameters are used which essentially try to construct a bounding burstiness function. A bounding burstiness function $b(t)$ of a trace satisfies the following inequality:

$$b(t) \geq \text{Maximum amount of traffic sent in any interval of length } t.$$

From the simple model one can reconstruct a bounding burstiness function which still bounds the source's traffic rate, though this bound would be loose as compared to the burstiness function of the source.

The leaky bucket model is a special case when the bounding burstiness function is represented using two parameters which bound the original burstiness function. A traffic source compliant to leaky bucket parameter (σ, ρ) is also compliant to bounding burstiness function as defined by:

$$b(t) = \sigma + \rho t$$

One can also construct models with more parameters which give a tighter bound for the burstiness function. For example. one can specify a series of σ and ρ such that the traffic satisfies each of the constraint imposed by the leaky bucket of parameters $(\sigma_i,\, \rho_i)$. Thus

$$b(t) \leq \min_i(\sigma_i, \rho_i t)$$

In this case $b(t)$ is piece wise linear and convex. In fact it can easily be shown that if the burstiness function of a traffic source is bounded by a continuous, convex and piecewise linear function, then there is an equivalent traffic descriptor in the form of sequence of leaky bucket descriptors, which completely describe the burstiness function. Each segment of the burstiness curve in this case, corresponds to a leaky bucket descriptor.

Unlike the burstiness function, the leaky bucket parameters of a trace are not unique. Any leaky bucket traffic descriptor (σ, ρ) such that the amount of traffic sent in any interval of length t is less than $\sigma + \rho t$ is a valid characterization of the trace. In general it is desirable to keep the parameters burst size and service rate as low as possible. As a result, there is a tradeoff in characterizing a traffic using leaky bucket. Reducing the service rate amounts to an increase in the bucket size and vice versa. This tradeoff can be seen from the burstiness function directly. For any traffic a leaky bucket traffic descriptor bounds the burstiness function by a straight line of slope ρ and y-intercept σ. For any burst size σ, a bounding tangent from point $(0, \sigma)$ to the burstiness function can be drawn as shown in Figure 2.1. The slope of the tangent gives the service rate for a given value of burst size. It is straightforward to see that as the burst size is increased, the service rate decreases. This tradeoff is later shown for various traces we consider in the leaky bucket traffic descriptor graphs.

Choosing appropriate descriptor is an important issue while characterizing the traffic using a leaky bucket descriptor. The definition of an appropriate descriptor depends on how the characterization is further used. If the characterization is used to perform admission control functions in the network, then the descriptor should maximize the number of connections accepted by the network. In order to do this, the values of σ and ρ should be as small as possible. As σ increases the service rate (ρ) decreases. The appropriate descriptor would lie in a region where the increase in the value of σ would not be justified (in terms of number of accepted connections in the

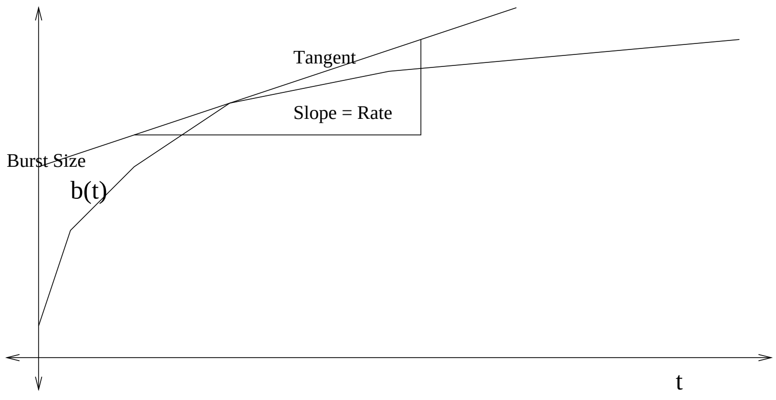

Figure 2.1: Tradeoff in σ and ρ as seen from burstiness function.

network) by corresponding decrease in the value of ρ and vice-versa. The slope of the graph would change very rapidly in this region (e.g. a knee point in the graph). This gives us an intuitive criterion to characterize the traffic using leaky bucket model. Such a characterization balances the two parameters σ and ρ and gives maximum network utilization.

Like the leaky bucket descriptor, the Tenet traffic descriptor also bounds the worst case rate. Therefore, it is possible to construct a bounding burstiness function such that a traffic stream compliant to the Tenet descriptor also complies with the burstiness function. The burstiness function of a traffic with descriptors $X_{min}, X_{avg}, I, S_{max}$ is given by the following equation:

$$b(t) = \left(\min\left(\left\lceil \frac{t \bmod I}{X_{min}} \right\rceil, \left\lceil \frac{I}{X_{avg}} \right\rceil\right) + \left\lceil \frac{t}{I} \right\rceil * \left\lceil \frac{I}{X_{avg}} \right\rceil\right) * S_{max}$$

Intuitively, the traffic source may initially send packets at spacing of X_{min} as long as the average rate criteria can be met. Since X_{avg} is calculated over an interval of length I, at most $\frac{I}{X_{avg}}$ packets can be sent and then the source will have to wait till time I. This pattern is repeated and gives the above expression for the burstiness function.

Although it is not possible to accurately reconstruct all the parameters of the X_{min}, X_{avg}, I model from the burstiness function, the important parameters X_{avg} and I and the tradeoff between them can be observed. Note that $b(I)$ is the maximum amount of data sent in any interval of length I. So, the worst case average rate over any interval of length I is $\frac{b(I)}{I}$. Therefore, the following expression relating X_{avg} and I may be used for analytic purposes:

$$X_{avg} = S_{max} \frac{I}{b(I)}$$

The burstiness curves plot $\frac{b(I)}{I}$ vs. I. $b(I)$ is the maximum amount of data sent in any interval of length I. Thus, $\frac{b(I)}{I}$ is the average bit rate in the interval in which the maximum amount of data is sent, which is the same as the worst-case average rate of the traffic taken over an interval of length I. The worst-case average rate for small intervals is close to the peak rate. As the rate increases, the worst-case average rate tends to the eventual average rate. Note that the characterization using the X_{min}, X_{avg}, I model is also not unique. It appears desirable to characterize the traffic using low average rate. This increases the averaging interval and as a result, the admission control procedure becomes less effective and ends up reserving bandwidth close to the peak rate. Thus a proper balance between parameters I and X_{avg} is needed.

The slope of the burstiness curve (i.e. the rate of decrease of the worst-case average rate with increase in the averaging interval) is a measure of the burstiness of the traffic at various time scales. The slope at an interval I measures the additional smoothing obtained when the bit rate is averaged over longer intervals, which characterizes the burstiness of the traffic at a time scale of I. If the traffic is not bursty in the short term, then averaging the bit rate over small intervals will not result in any smoothing. As a result, the slope of the burstiness curve will be zero for small intervals. If the traffic is not bursty in the long term, then all the smoothing would have happened when the bit rate is averaged over a long enough interval. Increasing the interval size will not result in any additional smoothing, and so the slope of the burstiness curve will again be zero. Thus, the slope of the burstiness curve characterizes the burstiness of the traffic at all time scales.

Let us illustrate this with an example. Figure 2.2 shows two sample traces of bit rates. In trace A the bit rate changes randomly with time, but the average bit rate over long period of time remains stationary. Thus, trace A is bursty over short time scale and is smooth over longer time scales. The bit rate of trace B remains constant

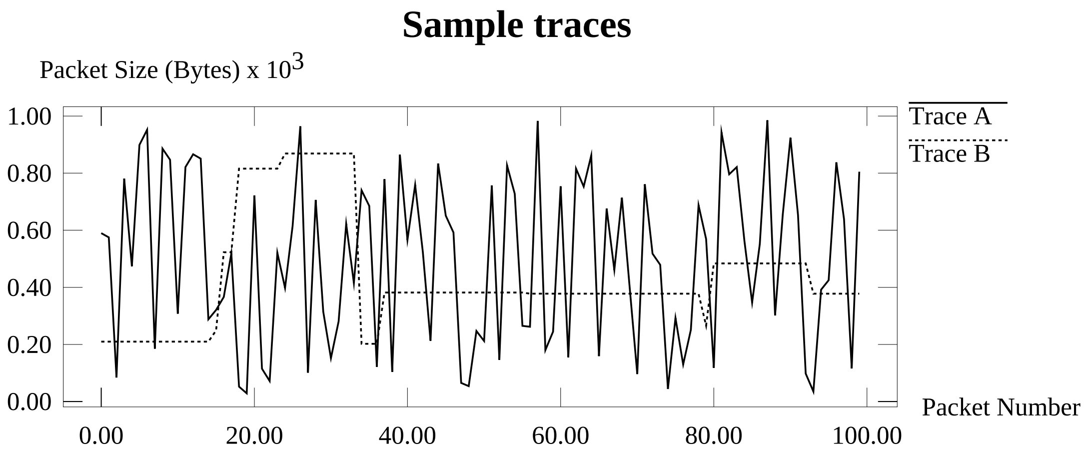

Figure 2.2: Bit rate of sample traces.

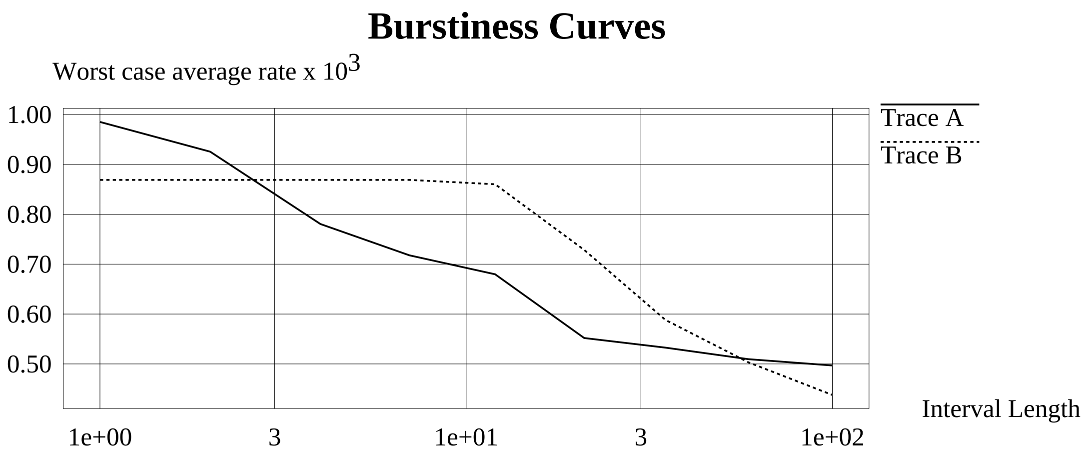

Figure 2.3: Burstiness curves for the sample traces.

when observed for short intervals of time. The changes in the bit rate occur after large time intervals and the bit rate is stable after these changes. Therefore, trace B has little burstiness in short term, and high burstiness in long term. This intuitive notion of burstiness is characterized by the burstiness functions of these traces as shown in Figure 2.3. For trace A, the slope of burstiness function is large for short intervals, and it tends to zero for long intervals. On the other hand, the slope of burstiness function for trace B is close to zero for small intervals and becomes large for larger intervals.

TRACE	Approx Duration	Frame rate (FPS)	Content
LECTURE	56 min	30	Lecture of a graduate course
NEWS	122 min	30	News program
MUSIC	122 min	30	A music program
SPORT	122 min	30	Basketball game
STWAR	114 min	25	Entertainment movie

Table 2.1: Traces of JPEG compressed video.

2.3 Methodology of Data Collection

2.3.1 JPEG Compressed Video

The data contains traces of five independent 1-2 hours long video clips, summarized in Table 2.1. The video clips were chosen to span a wide variety of video types ranging from a lecture in a class to a basketball game.

The video corresponding to the trace LECTURE was a videotape of a graduate course lecture. Most of the time, the video shows the instructor standing next to a board. It would occasionally show a student asking questions. The traces NEWS, MUSIC and SPORT correspond to real life video broadcast on popular TV channels. These clips were chosen to span a variety of programs, possibly of different natures. Since these videos are taken from TV channels, they are true representatives of a large majority of real-life video sequences. The trace STWAR is a trace of the movie "Star Wars".

All the data except STWAR was collected in real-time using DEC's JVIDEO board, which is able to carry out the JPEG compression of the 232 x 320 pixel colored images at the frame rate of 30 frames per seconds (fps). The compression was carried out at the minimum possible quantization factor to get the best picture quality[1]. The

[1]It is hard to define "constant" or "good" picture quality because it involves subjective perception. We use the quantization factor as a crude measure of quality.

TRACE	Approx. Duration	Frame rate (FPS)	Content
LECTURE.mpg	60 min	15	Lecture of a graduate course
NEWS.mpg	58 min	15	News program

Table 2.2: Traces of MPEG compressed video.

quantization factor was kept constant in order to get a constant picture quality[2]. The compressed data was discarded, only the frame sizes were recorded. The process of capturing the data was timed to verify that the capture rate was actually 30 fps.

2.3.2 MPEG Compressed Video

The video clips were first compressed in JPEG and stored on a local disk in real-time. The disk was just able to keep up at a frame rate of 15 fps. Then this data was compressed off-line into MPEG using a software encoder[3]. For various reasons, 1-5% of the frames were dropped before the final traces were obtained. The parameters of the MPEG encoder were chosen to minimize the running time of the encoder and to get a good picture quality. The quantization factors were kept constant to get a constant picture quality[4]. Table 2.2 summarizes the information about the MPEG traces. In the table X.mpg is the MPEG compressed version of the video X.

2.3.3 Traces of data sent by NV

The NV traces were collected using the *tcpdump* program. The traces are of data sent for MBONE[5] [30] video-conferences held on XUNET[6] [40]. There was an active

[2]Default quantization matrices were used for the compression.

[3]We used the Berkeley software MPEG encoder to carry out the MPEG compressions.

[4]The encoded stream was an MPEG I stream with quantization factors of 2, 5 and 15 for I, P and B-frames respectively. Default quantization matrices were used.

[5]MBONE stands for Multicast backBONE, which is a virtual network on the Internet created to experiment with multicast.

[6]XUNET stands for eXperimental University NETwork. It is a wide-area network that serves as a test-bed for research on data communication techniques.

TRACE	Approx. duration	Date & time of conference	Sender
NV0	17 min	09/22/93, 12:05pm	law.cs.berkeley.edu
NV1	74 min	10/20/93, 11:19am	law.cs.berkeley.edu
NV2	58 min	10/20/93, 11:19am	law.cs.berkeley.edu

Table 2.3: Traces of traffic generated by NV.

participant on the same LAN segment on which the observations were taken. During data collection, traces of all the packets on the network were captured. The fraction of packets dropped was extremely small (176 packets out of 940695, for instance). Packets not originating from the local participant were filtered out off-line. In all the traces, since the sender was in the same LAN segment, the traffic characteristics of the traces are expected to be close to the source traffic characteristics of the NV program. Table 2.3 summarizes the information about the traces.

In the trace NV1 it was found that, after 58 minutes from the beginning, the average bit rate went up to a value significantly higher than the routine 128 Kbps and after three minutes, the bit rate came back to normal. Probably the participant turned up the rate control knob of the NV software, and after 3 minutes he realized his mistake and became "nice" to the network again. NV2 consists of the first 58 minutes of the NV1 trace.

The traces of NV are packet traces whereas the JPEG and MPEG traces are frame traces. Thus, the NV traces are also affected by factors other than its coding algorithm. These factors include rate control built in the software, the rate at which the software can send packets into the network (limited by ethernet capacity in this case) etc. It is still important to consider these traces because NV is one of the very few applications that actually use the network to send live video. Studying these traces would also help us understand the interaction of system level parameters with traffic characterization.

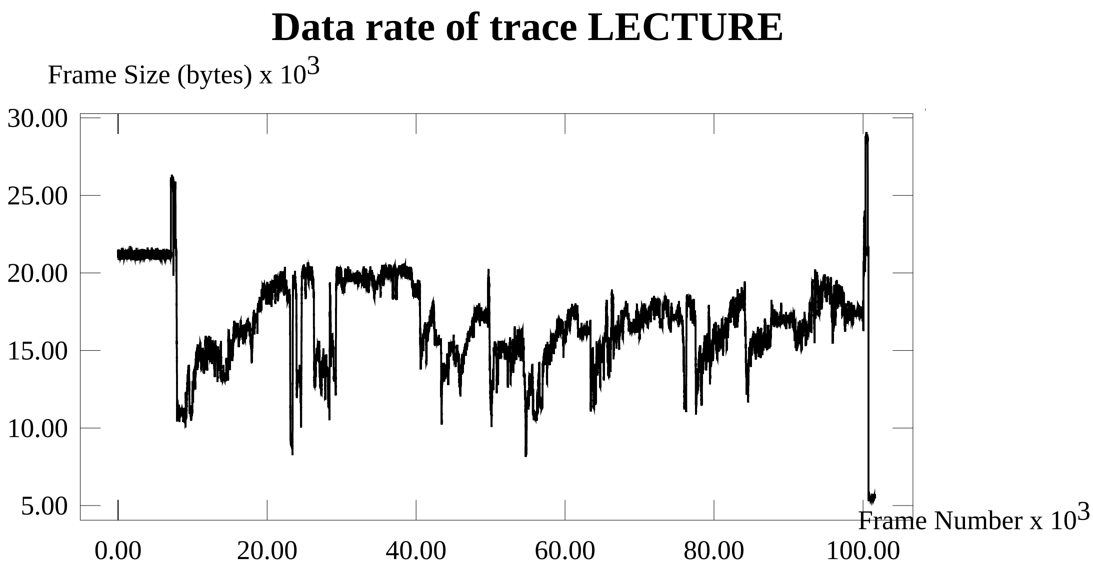

Figure 2.4: Bit rate of the trace LECTURE.

2.4 Characteristics of JPEG traffic

Figure 2.4 shows the bit rate of the trace LECTURE. The first ten minutes of the corresponding video shows the clock tower of UC Berkeley. The bit rate remains fairly constant during this period. The bit rate then shoots up as the text describing the lecture is superimposed on the clock tower. The lecture then begins with the instructor standing in front of a clean board. The time at which the lecture begins can be precisely identified by a sudden drop in the corresponding bit rate. The bit rate then increases steadily as more and more text is written on the board. The bit rate drops down again when the instructor starts writing on the second board and once again the video shows the instructor next to a clean board. The lecture goes on . . .

Figure 2.5 shows the bit rate of a 130 second clip of the trace NEWS. It is interesting to note that the bit rate remains nearly the same for some time, then it suddenly changes to a new value, it stays at its new value for some time and then this behavior is repeated. Analysis of actual video streams shows that in most of the cases these abrupt changes in bit rate correspond to scene changes (or cuts) in the video. For a scene showing the same people with the same background, the bit rate

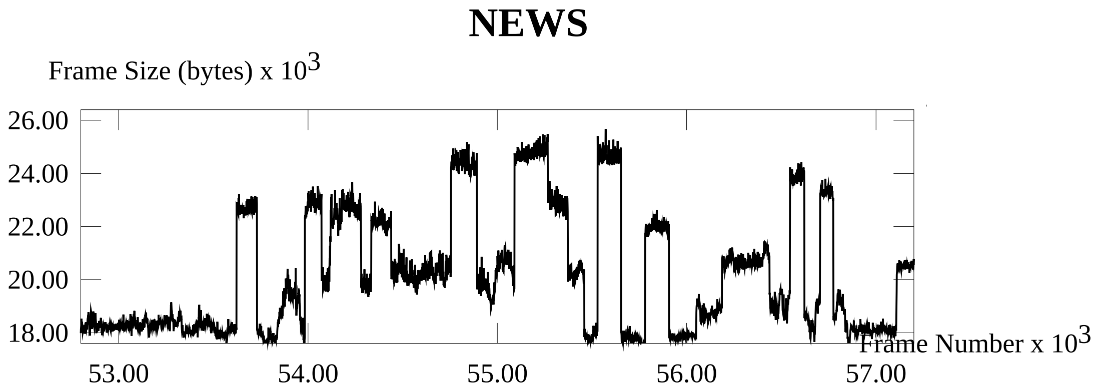

Figure 2.5: Bit rate of the trace NEWS.

remains nearly the same. As the background and orientation of people change, the bit rate changes slowly.

From visual inspection of the bit rates, it is apparent that the frame sizes of consecutive frames are highly correlated. The reason for this behavior is that the size of a compressed frame is essentially determined by the complexity (information content) of the image being compressed, and consecutive images of a video sequence are very similar in visual appearance as well as in their information contents. This similarity remains for longer runs of consecutive frames. As a result the bit rate is highly correlated over that period. In the trace LECTURE, the bit rate increases as the instructor writes more and more text on the board. As more text is written on the board, there is more information in the corresponding image. This results in a lower compression and hence a higher bit rate. Scene changes mean an abrupt transition from one sequence of pictures to another. The two pictures at the boundary of a scene change are not similar to each other, so the sizes of the corresponding compressed frames are also uncorrelated. Therefore the bit rate changes abruptly at the boundary of scene changes in the video.

The peak bit rate of a trace corresponds to a scene of high information content in the corresponding video. The bit-rate remains close to the peak rate for the duration of the scene, which will usually be of the order of seconds. Thus, the worst-case average bit rate over an interval of length smaller than the peak scene length will be close to the peak rate. The burstiness curves in Figure 2.6 substantiate this claim. Figure 2.6 plots the worst-case average bit rate of the traffic taken over an interval against the interval length. If the interval length is small, then the worst case average rate will be close to the peak rate. In the trace LECTURE, the worst-case average rate remains close to the peak rate for intervals of length up to 10 seconds. This is the approximate duration of the scene showing the clock tower along with the text describing the lecture. For other traces this interval is smaller. As the interval length increases, the worst-case average rate comes closer to the eventual average rate. For an interval of length equal to the length of the trace, the worst-case average rate is equal to the eventual average rate. It is interesting to note that the graphs in Figure 2.6 are very similar to each other for different video sequences. All the graphs start off with zero slope, which remains close to zero for intervals of length 300 ms to 10 seconds. As the interval length increases, the magnitude of the slope increases. Towards the end, the slope converges to a small negative value.

The same information is depicted numerically in Table 2.4, where *average* corresponds to the eventual average rate (in megabits per second) of the compressed video. All the traces of JPEG compressed video have average rates in the range 5-6.7 Mbps. The column labelled *burstiness* corresponds to the ratio of the peak rate to the eventual average rate, which is between 1.56 and 2.82. The peak bit rate varies from 19 Mbps to 7 Mbps. The subsequent columns list the ratio of the worst-case average rate taken over an interval to the (eventual) average rate for different intervals. As the interval size increases, this ratio becomes closer to 1. It should be noted that this ratio for intervals of length 100 ms is close to the peak to mean ratio of the stream. These numbers behave in a similar way for each of the four different traces of JPEG compressed video. For example, the ratios of the worst case average rate over an interval of 10 min to the average rate for different traces are between 1.00 and 1.25.

Figure 2.7 plots the leaky bucket traffic descriptor graphs for the traces. For each trace, the service rate ρ varies from a little more than peak rate to the eventual average rate. For each service rate, a bucket size is calculated by running the trace through a leaky bucket simulation. This bucket size is plotted in the graph. An interesting observation is about the knee points in these plots. In all of these curves,

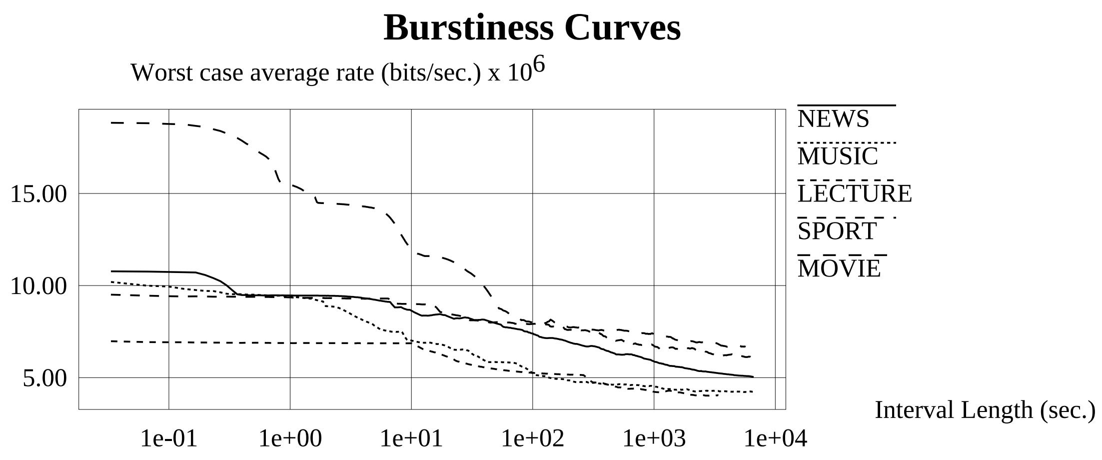

Figure 2.6: Burstiness curves for JPEG compressed video.

Trace	Average (Mbps)	Burstiness	Worst case average rates / Eventual average rate I = 100 ms	I = 1 sec.	I = 100 sec.	I = 10 min
NEWS	5.04	2.13	2.13	1.88	1.47	1.24
MUSIC	4.23	2.41	2.35	2.22	1.24	1.09
LECTURE	4.02	1.73	1.72	1.71	1.31	1.09
SPORT	6.09	1.56	1.54	1.54	1.30	1.14
MOVIE	6.67	2.82	2.81	2.32	1.20	1.13
LECTURE.mpg	0.61	9.88	5.53	1.95	1.24	1.15
NEWS.mpg	0.51	11.84	6.74	3.54	1.88	1.20
NV0	0.127	43.48	31.16	2.82	1.03	1.01
NV1	0.121	47.54	33.12	9.03	5.60	2.06
NV2	0.101	48.22	35.37	3.64	1.26	1.12

Table 2.4: Effect of I on the worst-case average rate.

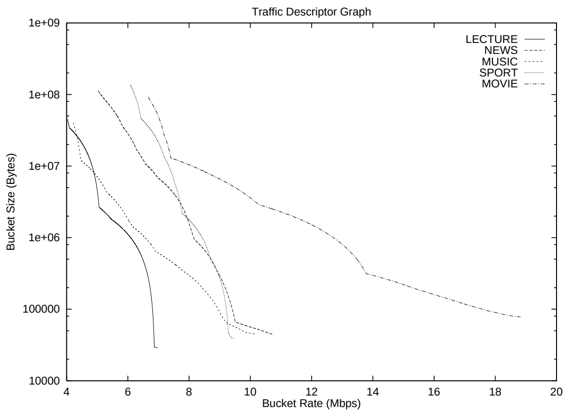

Figure 2.7: Leaky bucket traffic descriptor graph for JPEG compressed video.

the position of the rightmost knee corresponds to the peak rate of the trace. The bucket size σ grows rapidly as soon as the service rate ρ becomes less than the peak rate. For example, for the trace LECTURE, if the service rate is 80% of the peak rate, then the bucket size becomes as large as 1.6 M-Bytes.

A reasonable characterization of JPEG compressed video traffic using a leaky bucket descriptor would require that the service rate be close to the peak rate. If one characterizes the video using a lower service rate, the bucket size required will be very large, which is not an accurate characterization of the actual traffic. A traffic characterized by a large bucket size can potentially inject a large burst of data in the network, whereas the behavior of JPEG compressed video traffic tends to be more regular. The large bucket size in this case is not because of any bursts in the traffic instead, it is because, the instantaneous arrival rate remains a little higher than the service rate for relatively long periods of time. The bursts are only due to the individual frames.

The service rate of leaky bucket for our JPEG traces was between 19 Mbps and 7 Mbps, and the bucket sizes varied from 30 Kbytes to 80 Kbytes.

For constant quality JPEG compressed video, the bit rate depends upon the complexity of the images in the video being compressed[7]. Thus, the peaks of the bit

[7]The property discussed here is not an inherent property of the algorithms proposed by JPEG standards. It is only due to the fact that the encoder used in our experiments compressed the images of individual frames at a constant picture quality.

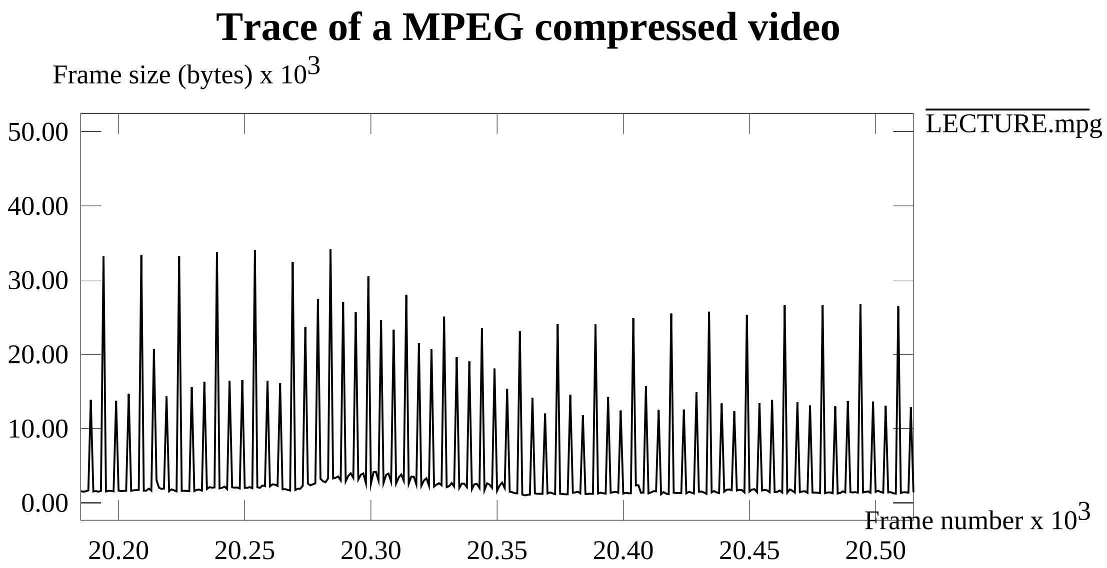

Figure 2.8: Bit rate of the trace LECTURE.mpg.

rate occur in bursts of length equal to the length of the scene having the high image complexity. This implies that, if a scene of high complexity is being transmitted over the network in real-time, then the network should have enough bandwidth available over the length of the scene to support the transfer. Queueing inside the network may not be able to accommodate high bit rate bursts. Thus, using a simple leaky bucket descriptor to characterize traffic would not yield any statistical multiplexing gains because it would require that the allocated rate be close to the peak bit rate of the actual video.

2.5 Characteristics of MPEG traffic

Figure 2.8 shows the bit rate of a 20 second clip of the MPEG trace LECTURE.mpg. The bit rate of this clip is very bursty. This burstiness is due to the three different types of frames generated by the MPEG compression algorithm. Notice that I-frames, P-frames and B-frames can be easily recognized in the figure. I-frames are the largest, and B-frames are the smallest in size. Notice that the sizes of frames of the same type are highly correlated.

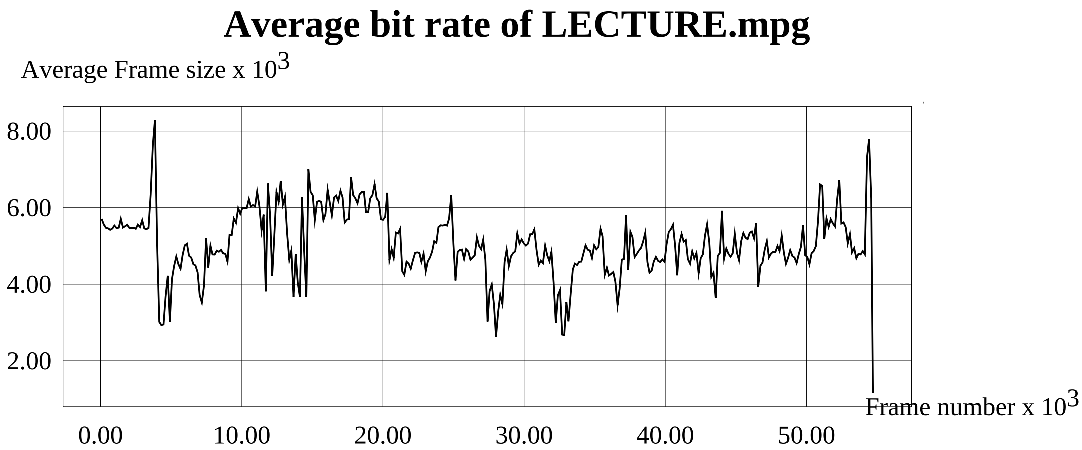

Figure 2.9: Average bit rate of the trace LECTURE.mpg.

Figure 2.9 shows the average bit rate of the trace LECTURE.mpg. The average is computed by a moving average filter of length 1 second. An interesting observation is the similarity of the graphs in Figures 2.4 and 2.9. The same lecture was encoded into JPEG and MPEG separately, and the variation of the average bit rate of both encodings appears to be very similar.

Figure 2.10 plots the burstiness curves for the MPEG traces. For small intervals, the worst-case average rate decreases quickly as the interval length increases. As the interval length becomes more than 1 second, the rate of decrease in the worst-case average rate becomes smaller. Table 2.4 shows that the average rates of the MPEG video streams are 0.61 and 0.51 Mbps respectively. From the same table we see that the *burstiness* values of the two traces are 9.88 and 11.84. The peak rates are about 6.0 Mbps. After averaging the bit rate over an interval of 1 second, the videos show burstiness of 1.95 and 3.54.

The graphs in Figure 2.10 are very similar to each other. Both have large negative slope for small intervals. The magnitude of the slope is smaller for interval lengths between 300 ms and 1 second. As the interval length increases further, the slopes converges to a small negative value.

MPEG encoded video shows high burstiness when observed over short time scales. This is due to the fact that the MPEG encoding algorithm generates different types

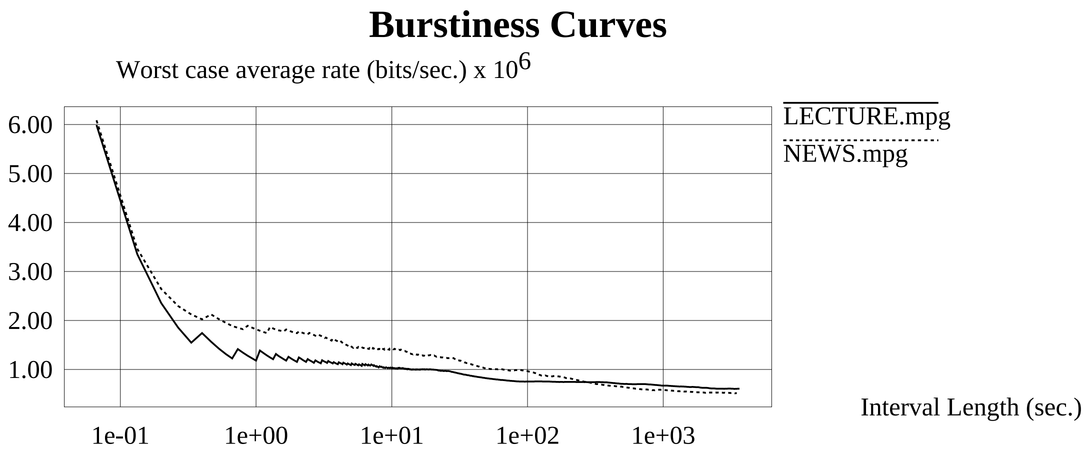

Figure 2.10: Burstiness function for MPEG compressed video.

of frames whose sizes are inherently different[8]. However, when averaged over an interval of length 1 second, the bit rate becomes smoother. This is because of the characteristics of the MPEG encoding algorithm. One of the considerations while designing MPEG was the ability to have random access to the frames. In order to achieve this, intra-coded frames (I-frames) are periodically present in the stream. Since I-frames are encoded independent of other frames, they provide random access points in the stream. P-frames and B-frames, which occur more frequently, constitute the rest of the video. As a result, MPEG video becomes a lot smoother if averaged over an interval which contains 2 or more I-frames.

After smoothing, the bit rate starts showing dependency on the contents of the video. This dependency can be clearly seen in Figure 2.9. Due to the periodic presence of I-frames in the stream, the bit rate of MPEG video depends upon the complexity of the pictures that are sent as I-frames. The inter-frame nature of the algorithm makes the bit rate dependent on the motion in the video. In case of the video LECTURE, the motion was relatively low. As a result, the bit rate averaged over periods covering multiple I-frames showed a high correlation with the rate of the corresponding JPEG trace.

[8]It is possible to have a valid MPEG stream which consists of only I-frames or only I-frames and P-frames. The presence of all the three types of frames in the stream makes the compression ratio higher.

Figure 2.11 plots the leaky bucket traffic descriptor graphs for the MPEG traces. The service rate ρ for each trace varies from peak rate to the eventual average rate. It is interesting to note that the bucket size (σ) does not increase even if the service rate (ρ) is less than the peak bit rate. There is a knee in the curve after which the bucket size starts increasing rapidly as the service rate decreases. The rate of increase is comparable to that in Figure 2.7. The knee occurs at a rate which is lower than the peak rate. This is so because the largest frame is of I-type, and it is always followed by a number of small B-type frames. By the time the frame next to the largest I-frame arrives, the bucket has enough space to accommodate it without necessarily serving at the peak rate. The knee in the curve occurs at a rate close to the worst-case average rate over an interval of length one second.

An appropriate characterization of MPEG video using the leaky bucket model corresponds to the knee in the graphs of Figure 2.11. The service rate at these knees correspond to the peak rate of the MPEG video obtained after smoothing its bit rate over one second intervals. The bit rate of MPEG compressed video becomes highly correlated after smoothing, and the changes in the bit rate are mostly due to the contents of the video. Therefore, the knee points characterize the traffic accurately. For the trace LECTURE.mpg, appropriate service rate and bucket size are 1.07 Mbps and 50 Kbytes respectively.

MPEG compressed video exhibits burstiness on two time scales. On a short time scale, the burstiness is due to the characteristics of the coding algorithm, which generates three types of frames of different average sizes. The short-term burstiness smoothes out when the bit rate is averaged over intervals of appropriate length[9]. This length depends upon the periodicity of I-frames in the stream. After this smoothing, the traffic remains bursty on a longer time scale. The long term burstiness for our streams was 1.95 and 3.54 respectively, due to the changes in motion and the complexity of scenes[10]. Since the smooth bit rate remains correlated for longer periods of

[9]It must be pointed out that this interval is not equal to the delay incurred in smoothing the bit stream. The maximum delay incurred in smoothing is given by $\frac{\sigma}{\rho}$. See Section 2.1 for details.

[10]The short-term burstiness property is an inherent property of the algorithms proposed by MPEG standards. However, it also depends on the encoding process. In our case, short-term burstiness

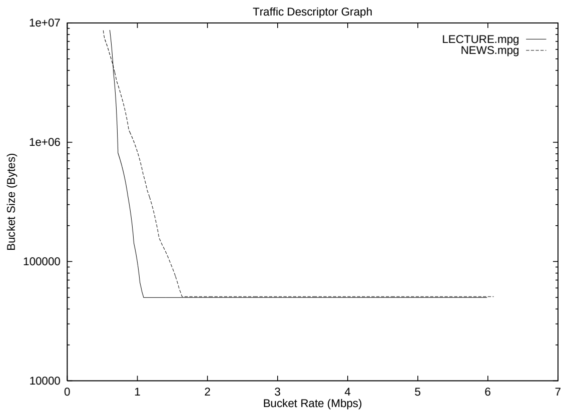

Figure 2.11: Leaky bucket traffic descriptor graph for MPEG compressed video.

time, additional buffering is unable to smooth the bit rate further. The knee point in the leaky bucket traffic descriptor graphs show the same fact. The amount of additional buffering needed to be able to serve at a rate lower than the rate of the knee is large. Therefore the knee points give a proper leaky bucket characterization of the video. Another implication of this is that, for transmission of MPEG video over a network, the available bandwidth of the network at any instant should be greater than the short-term average rate of the MPEG stream. In case this bandwidth is not available, congestion would result in a series of localized packet losses. The size of the leaky bucket can also act as a measure of the buffering needed to smooth the short-term burstiness.

2.6 Characteristics of NV traffic

Characteristics of the traffic generated by NV are highly influenced by the rate control built in the software. Figure 2.12 shows the inter-packet spacing for a series of packets generated by NV. Most of the packets are spaced at 5 ms, which is probably the time it takes to send a packet. The Inter-packet spacing of some packets periodically becomes as large as 0.5-2 seconds. This is due to the fact that NV stops sending data for long intervals if when realizes that it is sending the data at too high a rate. So,

---

also depends upon the ratio of the quantization factors of I, P and B-frames.

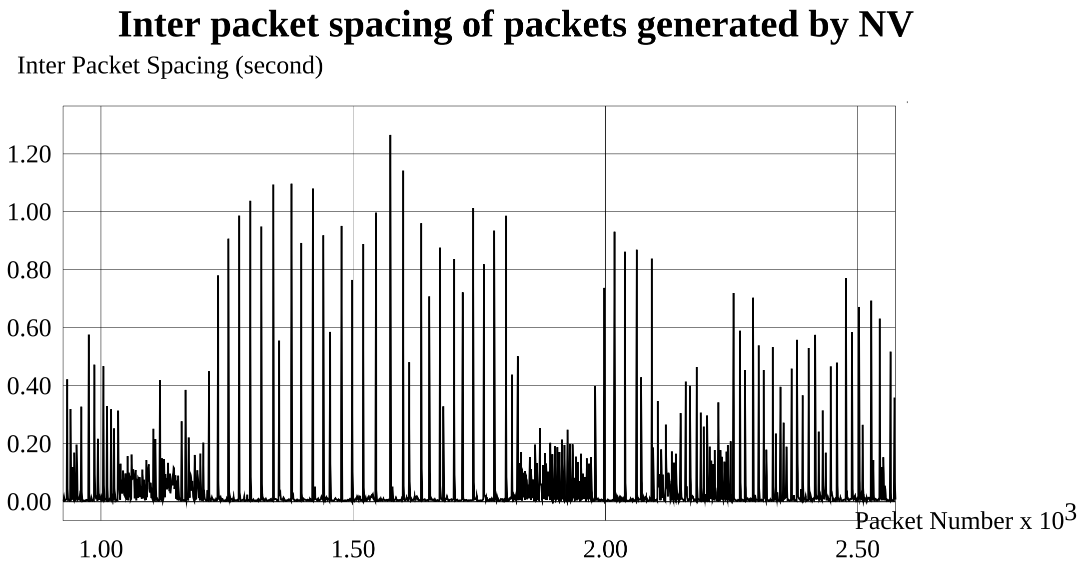

Figure 2.12: Inter packet spacing of packets produced by NV.

there are periods of activity when NV sends the data at high rate, and then there are periods of inactivity, when NV "sleeps" to lower the mean bit rate. Another observation in Figure 2.12 is that there are intervals during which the inter-packet spacing becomes as high as 250 ms. This can be attributed to the periodic waking up of other processes on the same host, or to other activity in the same LAN segment. Since the NV program carries out the compression of video images in software, it is a heavy user of the CPU which it shares with other processes. If other processes on the same system become active, then NV takes a longer time to produce a packet, and hence the inter-packet spacing can increase. This can also result from some periodic network activity on the same LAN segment (e.g., routing updates), which would force NV to wait to send data on the Ethernet.

The packet sizes are distributed between 0 and 1 Kbytes and the minimum inter packet spacing is as small as 0.2 ms. So the peak rate of the traffic generated by NV is very high. However, at averaging intervals of 1 sec, the worst-case average rate becomes close to the eventual average rate.

Figure 2.13 plots the burstiness curves for the NV traces. The peak rate is close

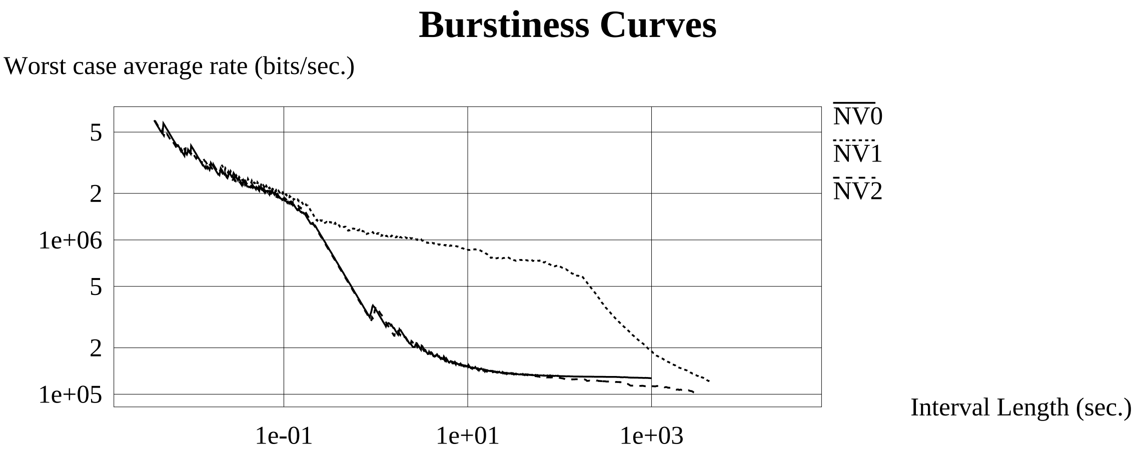

Figure 2.13: Burstiness curves for NV traffic.

to 6 Mbps, which is within a factor of two of the maximum available bit rate on an Ethernet. The worst-case average rate is bounded by 130 Kbps for averaging intervals of 10 seconds, which is because of the rate control built in the software. For the traces NV0 and NV2, the worst case average rate decreases rapidly with interval length till the interval length is close to 3 seconds. For interval lengths between 3 and 10 seconds, the rate of change of the worst-case average rate decreases. At intervals of length 10 seconds, the worst-case average rate becomes close to the eventual average rate. It is interesting to note that, although NV0 and NV2 correspond to two independent traces, their burstiness curves are nearly identical. The burstiness curves have a high slope for small intervals but this becomes very close to zero as the interval length increases to more than 10 seconds.

In Table 2.4 we see that the ratios of the worst-case average rate taken over intervals of length 100 sec to the eventual average rate for the traces NV0 and NV2 are 1.03 and 1.26 respectively. Trace NV1 is different because the target sending rate of the software was changed for 3 minutes. In Figure 2.13 there is a knee in the graph for NV1. This knee corresponds roughly to an interval of length 3 minutes.

Figure 2.14 plots the leaky bucket traffic descriptor graphs for the NV traces. When the service rate is higher than the peak rate, the bucket size is c.lose to 1 Kbytes. As expected, the bucket size increases as the service rate decreases. Traces

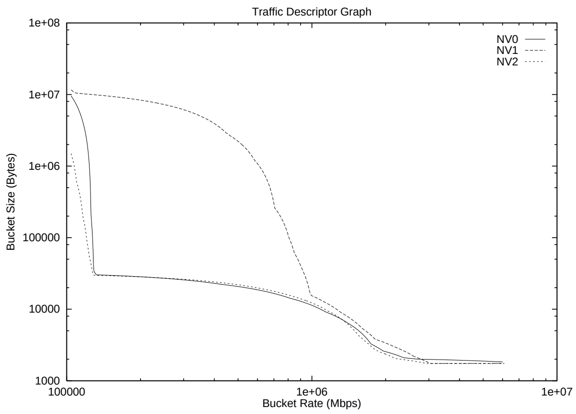

Figure 2.14: Leaky bucket traffic descriptor graph for NV traffic.

NV0 and NV2 have a knee in the graphs at a service rate of 130 Kbps. Trace NV1 has a knee in the corresponding graph at 1 Mbps. The position of this knee gives a reasonable characterization of the traffic using the leaky bucket descriptor.

The traffic generated by the NV program is highly bursty in the short term due to its on-off nature. The long term behavior of NV traffic is highly predictable. A plot of the average bit rate taken over 10 second intervals is nearly constant. Although the traces NV0 and NV2 are from two different conferences, their characteristics are nearly identical. This suggests that the characteristics of the traffic generated by NV are fairly independent of the video which is being sent. Constant bit rate over a long time scale is achieved by compromising the quality of the video being sent. Whenever NV needs to reduce its bit rate, it reduces the frame rate of the video being sent. For an NV program sending at a target rate of 128 Kbps, the appropriate service rate is 130 Kbps, and the bucket size is 30 Kbytes.

2.7 Choosing Leaky Bucket Traffic Descriptors

Choosing a traffic descriptor for a traffic is equivalent to picking up a point in the leaky bucket traffic descriptor graph of the trace. An optimal traffic descriptor should maximize the number of connections of given QoS. If each connection has different traffic descriptor and different QoS requirements, then the optimal traffic descriptor

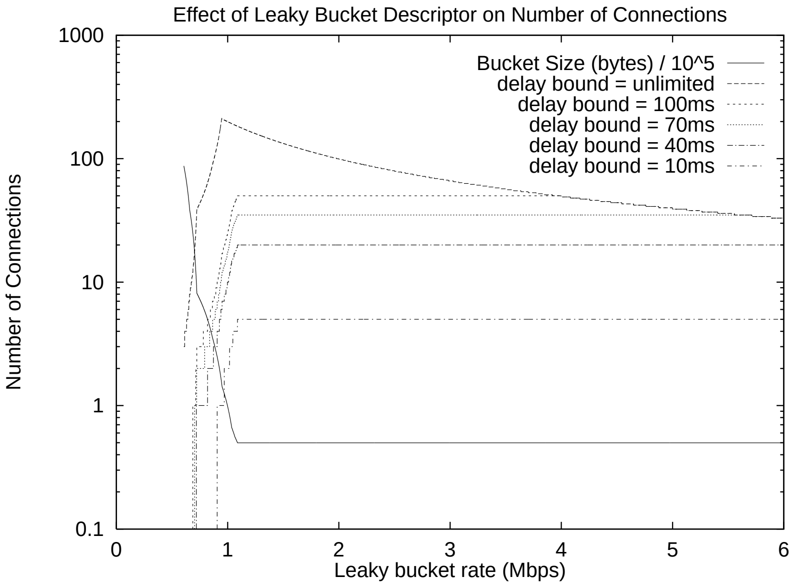

Figure 2.15: Trace LECTURE.mpg.

not only depends upon the properties of traffic being analyzed, but also depends, in general, on the QoS and the properties of the other traffic being multiplexed on the same link. To simplify, we assume that all connections are homogeneous. They have identical traffic descriptor and QoS parameters. Even with this simplification the problem of choosing a good traffic descriptor is not easy. A traffic descriptor characterizing a trace may be optimal if the connection requires a particular QoS, but the same traffic descriptor may be sub-optimal for a different QoS. In other words, an optimal traffic descriptor depends upon QoS. For most of the traces analyzed, we found that a traffic trace can be characterized by a single traffic descriptor which remains reasonably close to the optimal descriptor for the range of QoS considered. There are important implications of this finding. The single leaky bucket traffic descriptor characterizing the trace, may be presented to the network during connections setup, irrespective of the QoS required by the connection.

In order to analyze the effect of choosing traffic descriptor on number of connections, we assume a network of a single node having a bounded buffer and a single output link. The parameters of the switch were taken from XUNET II switch [40]. The outgoing link was assumed to have rate of 200 Mbps. The total buffer space was

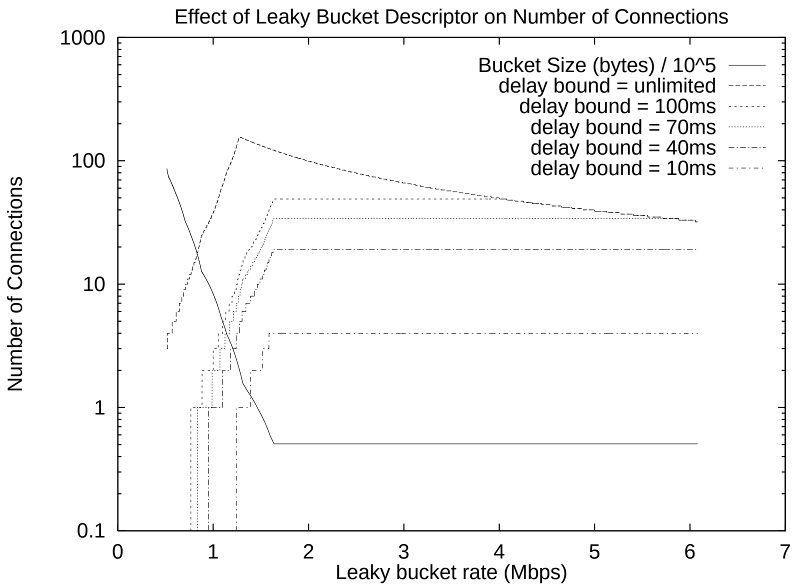

Figure 2.16: Trace NEWS.mpg.

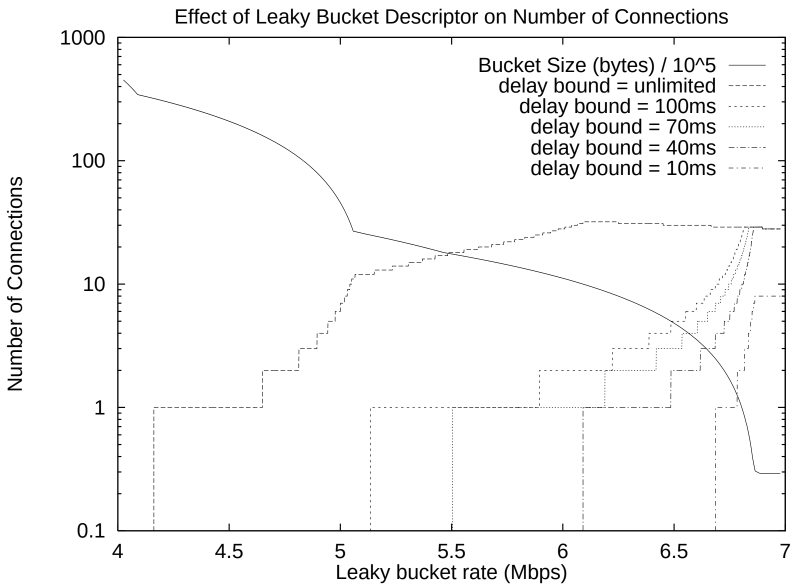

Figure 2.17: Trace LECTURE.

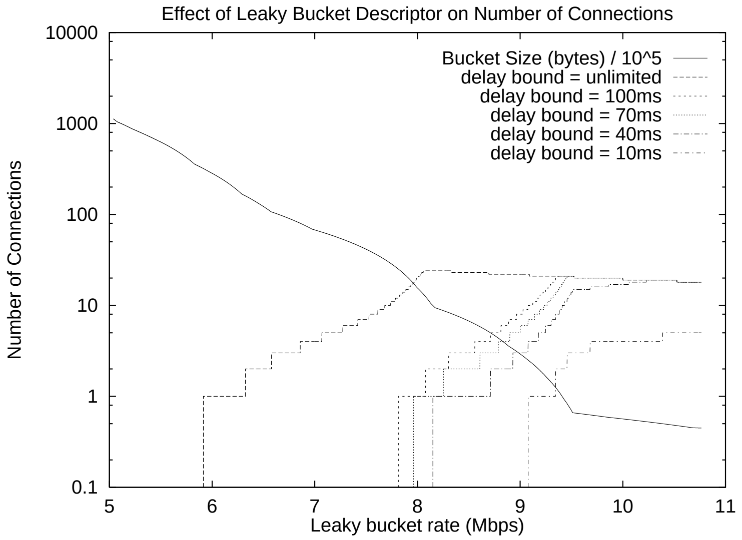

Figure 2.18: Trace NEWS.

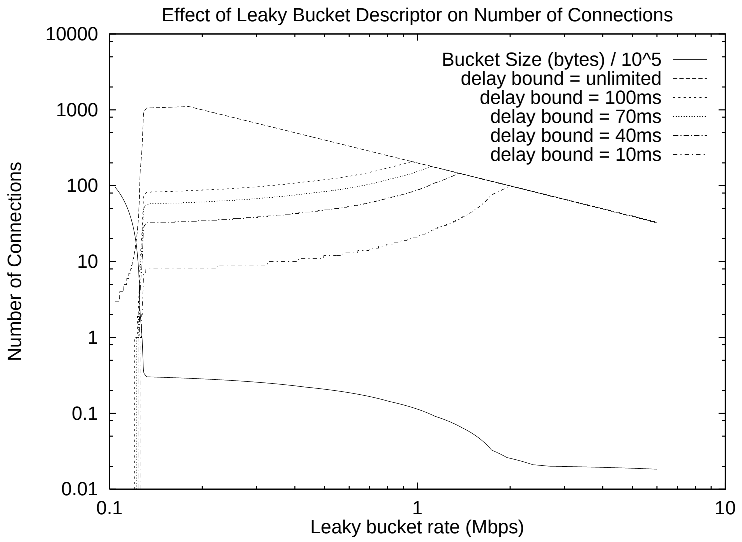

Figure 2.19: Trace NV0.

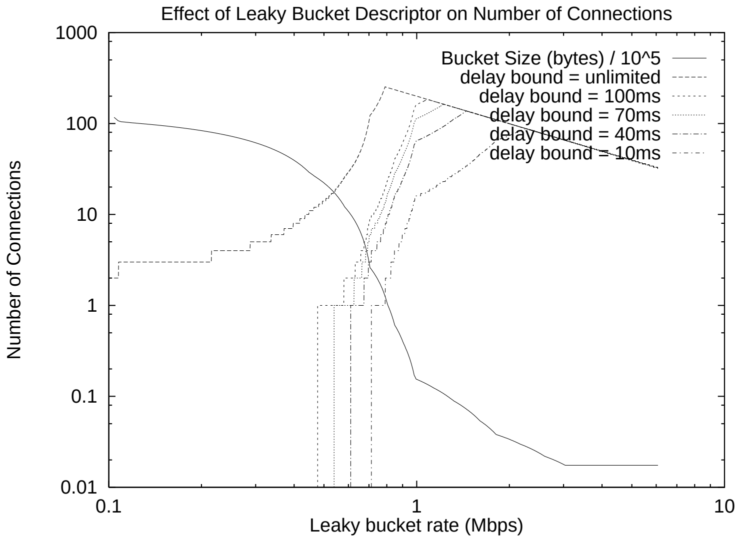

Figure 2.20: Trace NV1.

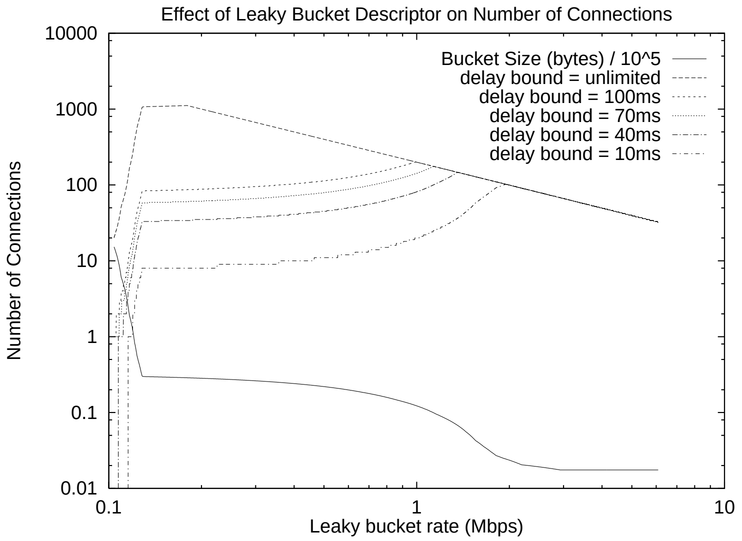

Figure 2.21: Trace NV2.

assumed to be 32 MBytes. We assume the optimum EDD scheduler at the output link and use exact admission control tests proposed in [84] for admission control. We assume homogeneous connections with identical traffic descriptor and QoS. The loss rate is assumed to be zero. Delay bound is the only variable QoS parameter.

Figure 2.17 shows the maximum number of connections accepted as a particular traffic descriptor is chosen to characterize the trace LECTURE.mpg. The X-axis represents the service rate of the descriptor, the Y-axis for the solid line represents the corresponding bucket size. The Y-axis for all other lines represent the maximum number of connections accepted.

If bounded delay is not required, the number of connections accepted is maximum. This is shown by the topmost line of the graph. The other lines show maximum number of connections under different delay constraints. Note the common property of all the graphs: the number of connections increase monotonically (starting from 0) as the bucket rate increases up to a point. After this, the number of connections start decreasing (with a break in slope) slowly and monotonically with increasing service rate.

When the service rate is low, the bucket size typically becomes large. Large bucket size results in failure of delay test while performing admission control. This limits the number of connections. Even if the delay bound is unlimited, the buffers required for each connection limits the number of connections. As the service rate increases, the bucket size decreases which relaxes both buffer and delay constraints. Therefore the number of permissible connections increases with increasing service rate. This trend continues till the aggregate bandwidth required is less than the link speed. After this, the link speed becomes the constraint and the number of connections decrease with increasing service rate.

It is interesting to note that the traffic descriptor which results in maximum number of connections is always close to the location of knee point in the traffic descriptor graph. For this trace, the knee point is at a service rate of 1.1Mbps where as service rate resulting in maximum number of connections for unlimited delay bound is 0.95Mbps. For delay bounds bound of 100ms, 70ms, 40ms, and 10ms service rate of 1.09Mbps gives maximum number of connections. Note that peak rate of this trace is 6.02Mbps and its mean rate is 0.68Mbps. This shows that even with large peak to average bit-rate ratio of the trace, the optimal traffic descriptors for different QoS are within a narrow range which is next to the knee point in the leaky bucket traffic descriptor graph of the trace. The same is true for the trace NEWS.mpg as shown in Figure 2.16. The location of knee is at service rate of 1.64Mbps, whereas the optimal traffic descriptors lie in the range 1.28Mbps to 1.64Mbps.

The characteristics of JPEG traffic is different from MPEG traffic. When the service rate is close to the average rate of the trace, the bucket size does not decrease significantly with increasing service rate. When the service rate approaches the peak rate, the rate of decrease in bucket size increases. This is evident from Figure 2.7. For the trace LECTURE, the effect of leaky bucket descriptor on number of possible connections is shown in Figure 2.15. For each delay bound considered, number of possible connections attain maximum value at service rate between 6.10Mbps and 6.90Mbps. This is very close to the position of knee, which occurs at a service rate of 6.86Mbps, in the leaky bucket traffic descriptor graph of the trace. Similarly, for the trace NEWS the number of connections achieve a maximum at service rate between 8.29Mbps and 10.70Mbps, whereas the knee in the traffic descriptor graph is at 9.50Mbps. This suggests that even for JPEG traffic, optimal traffic descriptors for different QoS are close to the position of the knee. The knee itself may be a good traffic descriptor for all QoS. All the other traces of JPEG compressed video are of similar nature.

The traffic trace NV0 is very bursty. Figure 2.19 shows how number of connections are affected by the choice of leaky bucket parameter. Unlike JPEG and MPEG traffic, where the optimal traffic descriptors for different QoS were found to be close to each other, in case of NV traffic, the optimal traffic descriptors for unlimited delay and delay bounds of 100ms, 70ms, 40ms and 10ms have service rates 180Kbps, 960Kbps, 1.13Mbps, 1.39Mbps and 2.98Mbps respectively. This makes it difficult to characterize the NV traffic optimally using a single traffic descriptor. There is a knee in the traffic descriptor graph at service rate 130Kbps. The maximum number of connections increase rapidly when the leaky bucket parameter is varied from average rate (128Kbps) to this knee. After the knee, the rate at which number of connections increase is lower. The trace NV2 has similar characteristics as that of NV0, but the trace NV1 is different. Its traffic descriptor graph has different characteristics. There is no well defined knee in the graph, but there are kinks at service rate of 0.98Mbps, 1.82Mbps and 3.00Mbps. The optimal traffic descriptors for this trace are in the range 790Kbps to 2.30Mbps which is narrower than that of NV0 and NV2, but still it is difficult to come up with a single traffic descriptor which gives close to optimal performance (in terms of maximum number of connections) for a wide range of QoS. Choosing leaky bucket traffic descriptor at second knee which is a service rate 3Mbps results in 66 connections which is within a factor 0.86 from the optimal for a delay bound of 10ms and within a factor 0.23 for unlimited delay bound.

Table 2.5 and 2.6 show a comparison of the maximum number of connections using optimal traffic descriptor and the knee point as the traffic descriptor. The table suggests that the knee point gives a good traffic descriptor for JPEG and MPEG traffic. For MPEG traces knee point gives optimal traffic descriptor in seven out of ten cases. In other two cases, the number of connections accepted using the suggested traffic descriptor is within 78% of the maximum possible. Only in one case the suggested traffic descriptor performs badly (57% of optimal). For JPEG traces, the suggested traffic descriptor is optimal in five out of ten cases whereas in other four cases it gives 73% of the optimal number of connections. Only in one case the suggested traffic descriptor gives 60% of the maximum possible number of connections. For NV traffic different traffic descriptors are needed for a trace depending on the required QoS. A single traffic descriptor is not sufficient for all QoS. While the suggested traffic descriptor for trace NV0 gives 91% of optimal number of connections for unlimited delay bound, it only gives 7% of the optimal for a delay bound of 10ms.

2.8 Discussion

The burstiness function characterizes the burstiness of the traffic at various time scales. JPEG video exhibits no burstiness on small time scales. This can be seen in the burstiness curves for JPEG traces by their zero slopes for small intervals. However, JPEG video is bursty over longer time scales. The burstiness curves for

Trace	Delay Bound (ms)	Number of Connections Maximum	Performance Using suggested TD	Ratio
LECTURE.mpg	Unlimited	211	181	0.85
	100	50	50	1.00
	70	35	20	0.57
	40	20	20	1.00
	10	5	5	1.00
NEWS.mpg	Unlimited	156	122	0.78
	100	49	49	1.00
	70	34	34	1.00
	40	19	19	1.00
	10	4	4	1.00
LECTURE	Unlimited	32	29	0.90
	100	32	29	0.90
	70	29	29	1.00
	40	29	29	1.00
	10	8	8	1.00
NEWS	Unlimited	24	21	0.87
	100	21	21	1.00
	70	21	21	1.00
	40	19	14	0.73
	10	5	3	0.60

Table 2.5: Comparison of optimal traffic descriptor and traffic descriptor at knee for JPEG and MPEG traffic.

Trace	Delay Bound (ms)	Number of Connections Maximum	Performance Using suggested TD	Ratio
NV0	Unlimited	1105	1016	0.91
	100	208	79	0.37
	70	177	55	0.31
	40	143	31	0.21
	10	98	7	0.07
NV1	Unlimited	253	66	0.23
	100	184	66	0.35
	70	161	66	0.40
	40	134	66	0.49
	10	86	66	0.86
NV2	Unlimited	1110	1072	0.96
	100	201	83	0.41
	70	176	58	0.32
	40	147	33	0.22
	10	102	8	0.07

Table 2.6: Comparison of optimal traffic descriptor and traffic descriptor at knee for NV traffic.

JPEG traces have a negative slope in the corresponding region.

MPEG video is bursty on short as well as long time scales. The burstiness curves for MPEG video have a high negative slope for small intervals. For larger intervals the magnitude of the slope is relatively small. This implies that MPEG video is more bursty on small time scales than on large time scales.

The traffic generated by NV is highly bursty over short periods of time, but is very smooth when averaged over an interval of 10 seconds or more. This is reflected in the burstiness curves by a large negative slope for small intervals and a zero slope for intervals greater than 10 seconds. The burstiness curve for the trace NV1 is different because, in the case of NV1, the target sending rate was increased for 3 minutes. As a result, NV1 has burstiness over a long time scale also. The burstiness curve for NV1 has a knee at an interval of length 3 minutes. The slope of the burstiness curve is steeper to the right of knee. This implies that NV1 is more bursty when observed over intervals of length a little more than 3 minutes, as compared with intervals of length slightly less than 3 minutes.

Leaky bucket traffic descriptor graphs for JPEG and MPEG video traffic have well defined knees. If the chosen service rate for leaky bucket is less than that of the knee point, then the bucket size becomes too large to be an accurate descriptor of the traffic. On the other hand, if the service rate is larger than that of the knee, then the reduction in the bucket size is small. The tradeoff of decreasing the bucket size by increasing the service rate above the service rate of the knee point may not be justified. Therefore, the knee points in the leaky bucket traffic descriptor graphs give appropriate leaky bucket parameters to characterize the video traffic irrespective of the requested QoS. In case of NV traffic, the knee points are not very well defined. In fact, the service rate of optimal traffic descriptor varies significantly with the QoS requested.

The appropriate service rate for JPEG compressed video is close to its peak rate. The service rate for MPEG compressed video is approximately equal to the peak rate of the MPEG stream after smoothing. The service rate of NV traffic is slightly more than its target sending rate if no delay bound is required. As delay bound for NV traffic is decreased, it needs higher service rate. For a delay bound of 10ms the required service rate can be as high as 1Mbps. For the trace NV1 the target sending rate was increased for 3 minutes. Thus, the service rate for NV1 is found to be higher (a little more than its maximum target sending rate) even with unbounded delay.

The bucket size parameter depends heavily upon the way the application delivers data to the network. In the case of JPEG and MPEG compressed video, we assume that the application delivers data in the form of frames, at a constant frame rate. Therefore, the bucket sizes obtained are close to the size of the largest frame present in the stream (30 - 80 Kbytes). If the application decides to break each frame into packets and send the packets uniformly over one frame time, then the bucket sizes will be different. The bucket size can be made arbitrarily small by suitably delaying the packets. The maximum delay incurred in this process is given by $\frac{\sigma}{\rho}$. For JPEG streams this delay was found to be 33 ms, which is equal to one frame time. This means that if the burst size of JPEG video is reduced, then the largest frame will be sent uniformly over one frame time. This delay for the MPEG trace LECTURE.mpg is 380 ms, which corresponds to about 6 frame times. For NV, the bucket size is between 5 and 10 Kbytes (equivalent to 3-6 packets of buffering) when the service rate is high. When the service rate is close to its target sending rate, a bucket size of about 30 Kbytes (sufficient to keep one frame worth of packets) is needed. The smoothing delays for NV is found to be 1.85 seconds, which is close to the measured off periods of NV.

2.9 Conclusions

Long traces of real video compressed using three different compression algorithms, namely JPEG, MPEG and NV, have been studied in this chapter. The chapter characterizes the traces using a leaky bucket model, and shows a way of choosing appropriate leaky bucket parameters. The burstiness function is used to characterize the burstiness of the traffic at various time scales. The impact of burstiness on network congestion is discussed.

It was found that the leaky bucket parameters for constant quality JPEG and MPEG compressed video depend upon the actual video streams. For JPEG streams, the appropriate service rate was found to be close to the peak bit rate of the compressed video. As a result, characterizing JPEG video using a leaky bucket descriptor is not expected to give any statistical multiplexing gains. The service rate of the JPEG compressed streams under study varied from 7 to 18 Mbps. The bucket sizes were found to be between 30 and 80 Kbytes. MPEG compressed video could be characterized with a service rate lower than its peak rate. The service rates for our traces were found to be 1.05 and 1.65 Mbps, which are 2-4 times the eventual average rate. The corresponding bucket sizes were approximately equal to 50 Kbytes. Due to the rate control built in the software, the traffic generated by NV can be characterized independent of its contents. The service rate depends upon the target sending rate set on the program. For a target rate of 128 Kbps, if the delay bound is set to unlimited, the required service rate was found to be 130 Kbps, and the bucket size was 30 Kbytes. The service rate increases as the delay bound is decreased. The service rate for a delay bound of 10ms is 2.3Mbps.

It was shown how burstiness curves can be used to characterize the burstiness of the traffic at different time scales. Using these burstiness curves, JPEG compressed video has shown to exhibit low burstiness over small time scales, whereas MPEG and NV traffic shows high burstiness. On long time scales, JPEG and MPEG video exhibit burstiness, whereas NV shows no burstiness.

Short-term burstiness, which is a property of the coding algorithms, is easier to handle. A traffic having short-term burstiness can be made smoother at the cost of additional delays. Even if the traffic is not smoothed, the worst-case buffer requirements to serve such a traffic at a constant bit rate are small. Long-term burstiness is more difficult to handle.

One way to handle long-term burstiness is by eliminating it in the encoder. Encoders are being designed which support constant bit rate operation. Since the bit rate depends upon motion and complexity of the video being encoded, such encoders will operate at a constant bit rate by reducing the picture quality of the scenes that have high motion or information content (for example, scenes showing a black-board full of text in a class room). On the other hand, the scenes which do not have much information (scenes showing a clean black-board) would be compressed at extremely good picture quality. This kind of behavior is certainly undesirable. One might want to limit the peak rate of the video to a high value by designing appropriate encoders, but, in normal mode of operation, one would like to have a video whose quality remains fairly constant.

Another approach for wide area networks might be to reserve the bandwidth at peak bit rate for connections carrying video. Policing in this case would involve peak rate enforcement, but finding the peak rate for live video in advance may be difficult. The problem can be solved if the encoder limits the peak rate of its output stream. The quality of scenes requiring higher bit rate can be compromised. The long-term burstiness is found to be between 2 and 4 for our traces. By reserving the bandwidth at the peak rate, the video connections can utilize the network up to 25-50%. If the non-real-time data traffic, which can adapt to congestion in the network, is able to take the rest of the available bandwidth, then higher network utilizations can be achieved.

One may want to reserve bandwidth somewhere between peak and average rate, and "hope" that statistical multiplexing among various video streams will smooth the aggregate bit rate. It could be hard to deal with congestion as all the clients can potentially send their data at peak rate. The multiplexing behavior of constant quality video needs to be studied in more detail.

Adapting the sending rate according to the feedback received from the network has also been proposed [69]. The network signals congestion to the application, and, based on this congestion signal, the application sending video adapts to a different rate by having a different compression ratio. This scheme may work well for interactive video, which encodes the video on the fly, but is difficult for stored video because changing the bit rate of stored video may involve additional processing. The additional complexity of this scheme may not be justified.

Chapter 3

Deterministic Guarantees: Performance Evaluation

In this chapter we discuss several approaches to providing Quality of Service (QoS) guarantees in an integrated service network. In particular, we focus on deterministic guarantees. We evaluate the performance of schemes providing deterministic performance guarantees to real-time network clients with variable bit-rate (VBR) traffic. Contrary to expectations, we found that deterministic guarantees can give good network utilizations. The primary reason for poor performance of deterministic guarantees was poor traffic models and sub-optimal admission control tests. We propose new and improved admission control tests for the EDF scheduler using a generic traffic model. Using these admission tests, we compare the performance of the X_{min}, X_{avg}, I traffic model initially proposed by Tenet Group at UC Berkeley with the leaky bucket model. Our studies supplement that leaky bucket model as specified in UNI signaling specifications of ATM-Forum is better than X_{min}, X_{avg}, I traffic model. We use long traces of video compressed using different algorithms for our analysis. We also compute the maximum achievable network utilization for real-time VBR only traffic. The maximum achievable network utilization for real-time VBR traffic is found to vary between 14.4 % to 70 %. If the fraction of CBR and ABR traffic is large then very high network utilization can be achieved. We conclude that if better admission control tests and suitable traffic models are used, then deterministic guarantees can give high network utilizations.

3.1 Introduction

Several approaches to provide QoS guarantees in an integrated service network have been proposed. The most important component of any approach is its service model. A service model defines various parameters related to the quality of service (QoS). It defines the parameters using which a user can express the desired QoS. For instance, in one service model a user may be allowed to numerically represent various parameters like bit rate, delay, delay jitter etc. As a result a user may be able to request a connection with any combination of these QoS parameters. In another service model a user may be restricted to choosing a QoS only from some predefined QoS classes. In this case the set of predefined classes is designed carefully to cater to all possible user needs which are expected to be important. This set may consist of classes like MPEG video, telephone audio, high quality audio, ordinary data and priority data.

The second component in providing QoS guarantees is the set of mechanisms needed to implement a service model. Most often, the choice of service model itself depends upon the ability to efficiently implement the underlying mechanisms needed to support it. For instance, due to the tremendous increase in communications bit rate in the past decade, packet switching is increasingly becoming a bottleneck in the communication path. Therefore service models which require significant overhead in the switching path are less desirable. Approaches which tend to add small overhead in the switching path are favored. For this reason, the ATM service model defines the transport of fixed size cells of 53 bytes through the network. Fixed cell size results in faster and more efficient switching.

This leads to a tradeoff in defining a service model. On one hand, the model should be generic enough to represent the QoS requirements of the current as well as future applications. On the other hand, the mechanisms needed to implement the service model should be simple and efficiently realizable in a large network. The decision whether a service model is sufficiently generic is very subjective. Even after many years of research, no consensus has emerged. From discussions, newsgroups and mailing lists it is evident that the debate on the choice of proper service model is going to continue in the research community. However, the mechanisms needed to realize a particular service model can be explored more scientifically. Various properties of these mechanisms like implementation complexity, speed, scaling with network size can be evaluated. We now list some important requirements which any implementation must meet.

3.2 Requirements

In a packet switching network, several communication links are interconnected through a switching node (often called a switch in the context of ATM networks or a router in the context of IP networks). A switching node forwards each incoming packet on a link to the appropriate output link depending upon the information stored in the packet header. In order to support QoS in an integrated service network the switching nodes need to carry out several other functions in addition to their basic function of packet forwarding. These functions can be broadly classified into *packet classification, packet scheduling* and *buffer management*.

A switching node needs to infer the QoS with which the packet should be treated based upon the information in the packet header. Before beginning data transmission, an application typically makes a resource reservation request in the network for its desired QoS. Data transmission begins only if the resource reservation request succeeds. The network stores the information about QoS and resource reservation in all the nodes in the data path of the application's traffic. When a packet arrives at a switching node, it finds out the amount of resources reserved for the traffic class to which the packet belongs, and services the packet accordingly. In ATM environment, a unique connection identifier called virtual connection identifier (VCI) is allocated for each connection at each switch. Each cell is tagged with the VCI of its connection. Thus the VCI can uniquely identify the connection to which the cell belongs. If the resources are reserved on a per connection basis, the VCI of a cell can directly give index into a table containing the desired information. This approach of direct table lookup indexed by VCI is feasible because the number of distinct VCIs at a switch is typically small (of the order of 10's of thousands) and these VCIs can be allocated in a contiguous manner. In IP environment, there is no notion of a connection in the routers. This makes the problem of packet classification harder. There is a notion of *flow* identified by the tuple (source IP address, source port, destination IP address, destination port). In order to identify the flow to which a packet belongs, its address and port number tuple must be matched with a list of all currently active flows. The IP address field is 32 bits in length and the port number field is 16 bits wide. Thus the total possible flows at a router can be as large as 2^{96}, which makes packet classification based on direct table lookup impossible. The complexity of packet classification is a serious drawback of the integrated services approach [20, 108, 110, 127] proposed by IETF to provide QoS in IP networks. To solve this problem, several newer schemes like differentiated services [22, 95, 94], IPV6 [25, 59] have been proposed that make use of a separate field in the packet header to mark its QoS.

Once a packet has been classified it has to be treated according to its QoS. Packet from low delay connections should be given priority over packets from high delay connections. Similarly a packet belonging to a connection requiring low loss rate should be given preference in allocating buffers. In case of buffer overflows, packets from connections with higher loss rate tolerance should be dropped first. This is done by using appropriate scheduling and buffer management policies at switching nodes.

Scheduling is an important function needed at a switching node. There is a scheduler at each output link of a node which decides the order in which packets from its output queue are transmitted. The scheduler ensures that each connection (or traffic class) gets its desired share of resources like bandwidth, delay etc. Since scheduling operation is needed for each packet, it is desirable to have low complexity schedulers. If the service model provides arbitrary delay bounds to individual connections or traffic classes, then complex schedulers like earliest deadline first (EDF) [47] or multi level priority [130] are required at each node. However, if there are limited and predefined QoS classes, then simple schedulers based on say few priority queues may be sufficient.

Buffer management schemes provide a control on the loss behavior of traffic. Due to the inherent burstiness of the traffic, the instantaneous arrival rate of traffic at a link may become larger than the link rate. As a result traffic is queued in the buffers at the link. These queues may become excessively large especially at points where several high speed links are connected to a low speed link. This leads to frequent buffer overflows. A buffer management scheme decides which packets must be dropped in case of buffer overflow. In addition to controlling the loss behavior of traffic with different QoS, a buffer management scheme can also enhance the overall network performance. For instance cell dropping schemes for enhancing the performance when TCP traffic is transported through an ATM network have been proposed in [107, 14, 67, 71, 121]. Similarly, schemes for packet dropping in IP routers for enhancing the performance of TCP have been proposed in [37, 85].

Another issue closely related to buffer management is packet marking. When traffic from a particular class exceeds its allocated rate, then instead of dropping the excess (or non conforming) packets, the switching node can simply mark the packets as non conforming. These non conforming packets are transported as long as bandwidth and buffer space is available in the network. In case of buffer overflow, the non conforming packets are among the first to be dropped. This ensures that if the network is lightly loaded, the applications can send data at high rate to increase network utilization. In case of congestion, the applications are guaranteed to get only their reserved share.

Amount of control state needed at switching nodes is increasingly becoming a scaling bottleneck in the network. The growth of networks has resulted in several millions of end users connected to the network. Each user can open multiple connections which pass through the same backbone in the network. If a service requires the switching nodes to maintain state for each connection passing through it, the amount of state information could very easily become unmanageable. Thus approaches are devised using which the amount of state information needed at each node is kept to its minimum. For instance, in ATM networks, using a common virtual path identifier, a bunch of connections are grouped together into a single virtual path. Intermediate switches only need to keep the information about the virtual paths, not of connections within the paths. Another approach using a VC-bunch to group several connections together has been proposed in [55]. In the differentiated services framework [22, 95, 94], which provides QoS guarantees in IP networks, care has been taken to make sure that the connection specific information is pushed to the edge of the network as far as possible. The intermediate routers do not need connection specific information.

As mentioned earlier an application desiring QoS reserves required network resources before beginning any data transfer. The resource may be reserved by the signaling protocol itself as in ATM networks [2, 19, 6]. Alternatively there could be a separate resource reservation protocol like RSVP [132] in IP networks. This protocol introduces overhead in network management. It is evident from the current trend in the Internet that individual connections tend to be short lived [3] and applications like web browsers open multiple parallel connections to reduce their latency [97]. Short connection lifetimes result in large number of connections to transfer a particular amount of data. This increases the overheads in connection setup and resource reservation. It is therefore desirable to have a little overhead in signalling and resource reservation. Resource reservation involves admission control :- a test to decide whether new connections can be admitted in the network without degrading the QoS of currently established connections. Admission control has to be done at each node in the path of the connection. Admission control at a node requires a computation involving traffic characteristics and QoS of all the connections passing through the node. A naive admission control scheme may take time proportional to the number of connections passing through the node. Smarter schemes aggregate the resource reservation information to reduce the complexity of admission control.

Finally, the overall network utilization achievable is a very important property of a service model. Generic service models tend to lower the network utilization as they need to accommodate a wider range of traffic mix. Also, models which are mathematically well defined are expected to have a lower utilization as compared to several other ad-hoc qualitative models.

3.3 A Survey of Service Models

Several service models have been proposed in the literature [60, 77, 80, 78, 79, 61, 34, 33, 20, 29, 133, 106, 17, 22, 95, 94]. These can be classified into four major categories: class based approaches, deterministic QoS guarantees, statistical QoS guarantees and ad-hoc assurances.

- Class based approaches: In these approaches an application can request for QoS only from a predefined set of QoS classes. This set should be generic enough to fulfill the requirements of each potential application. The set contains classes like MPEG video, audio, telnet traffic, world wide web traffic etc. The concept of schedulable region is defined for admission control. If there are N classes, then schedulable region is defined as a region in N dimensional space. Each axis in N dimensions represent number of connections of a particular QoS class. A point $(n_1, n_2, \ldots, n_N)$ is in schedulable region if and only if it is possible to admit n_1 connections of QoS class 1, n_2 connections of QoS class 2 and so on, such that each connection gets its QoS. Schedulable region is estimated empirically. Admission control test is equivalent to finding out if a point belongs to the schedulable region.

- Deterministic guarantees provide mathematically provable bounds on QoS parameters like delay, delay-jitter and loss rate. In a deterministic service model an application can ask for arbitrary (but realistic) delay bound for its traffic. If the QoS request succeeds, the network makes sure that the delay incurred in transporting any packet is always less than the delay bound. This property of delay bound can be proved mathematically. In a similar fashion an application can request for a bound on delay-jitter and loss rate.

  There are two important observations regarding deterministic guarantees. Firstly the QoS guarantees are quantitative and there is a mathematical proof of the guarantees. Moreover, whether the network is honoring the QoS commitments can easily be checked by monitoring the traffic and its delay and loss pattern. Secondly, the guarantees bound the worst case behavior like the worst cases delay, worst case loss rate. As a results, the deterministic guarantees are very conservative. Consider a set of N identical on-off sources. Let the on period be T and the off period be $2T$. Assume that these sources begin their cycle at random times. Now the probability that two sources are on at same time is less than half. Therefore the probability that all the N sources are simultaneously on is less than 2^{-N}. This number becomes almost equal to zero for large values of N. In practice, although it is highly unlikely that all sources are on at the same time, but it is still a possibility. Therefore, while reserving bandwidth, deterministic approaches will need a rate of Nr (where r is the sending rate of individual source). Because of this conservative approach network utilization for deterministic guarantees is expected to be low. This was a major criticism of deterministic guarantees. We will show later in this chapter that with careful admission control and traffic characterization, higher network utilizations can be achieved on a network providing deterministic guarantees. The traffic models used in earlier studies were very simplistic like peak rate [15], jumping window, moving window [105], and the Tenet traffic model [34, 33]. We show that with leaky bucket characterization better utilizations can be achieved. Earlier approaches used peak rate based admission control. Later some studies improved admission control tests for first-come-first-serve scheduling algorithm [35]. Deterministic guarantees can be implemented using a variety of scheduling disciplines like first-come-first-serve, static-priorities, earliest-deadline-first, weighted-fair-queueing and its variants etc.

- Statistical Guarantees: Instead of bounding worst case parameters, statistical QoS guarantees provide a bound on percentiles. In many cases, instead of bounding worst case delay, it is sufficient to ensure that the expected fraction of packets exceeding the delay bound is very small (say 10^{-7}). These guarantees are based on a statistical model of the traffic. A simple and widely used traffic model is the on-off source [54]. Statistical bounds obtained using the on-off sources have shown to be an upper bound in a fairly generic setting. Some of the work characterizes the traffic as Markov modulated autoregressive process and proposes admission control based on this model [56, 113, 88, 58]. Recently it has been shown that a variety of traffic has long range dependency and can be modeled only by a self similar process [102, 83, 46, 103].

Admission control tests have been worked out for on-off sources and Markov modulated sources. Some of these tests are based on the concept of calculation of effective bandwidth of the source [29]. These tests are based on the assumption that traffic sources can be modeled as a stochastic process with some “nice” properties. These assumptions may not be valid in real practice. There are other admission control schemes that are based on the theory of large deviations [133, 106, 17]. These do not make any assumption about the traffic sources except that the bit rate distribution is stationary. Based on this and the empirical distribution of bit rate, and theory of large deviations, admission control tests have been derived. Queueing analysis of long range dependent and self similar traffic is very difficult. Studies of their queueing behavior has been only limited so far [49].

Most of the studies of statistical queueing and loss behavior of traffic are limited for a first come first serve scheduler. The studies for other schedulers are also very limited.

Statistical guarantees may result in better network utilization as compared to deterministic guarantees. However, because of their statistical nature, it is difficult to measure the validity of statistical guarantees. Similarly it is difficult to police a traffic based on a statistical model.

- Ad-hoc Guarantees: An architecture with the intent of minimizing implementation complexity in a large datagram network have been proposed in [22, 95, 94]. This architecture achieves scalability by aggregating traffic classification state which is conveyed by means of packet marking using a special field in the packet header. Packets are classified and marked to receive a particular per-hop forwarding behavior on nodes along their path. Sophisticated classification, marking, policing, and shaping operations are only implemented at network boundaries or hosts. Network resources are allocated to traffic streams by service provisioning policies which govern how traffic is marked and conditioned upon entry to a such a network, and how that traffic is forwarded within that network. QoS is provided to aggregate traffic in a particular class. There is a service level agreement between a customer and a service provider which specifies the forwarding service a customer should receive. Similarly there is a traffic conditioning agreement which specifies the amount of traffic (of each class) which a customer is expected to generate. The lifetime of these agreements are long (like a month or more). The service provider provisions its network to make sure that the service level agreements of its customers are honoured. This architecture achieves scalability by pushing as much complexity to the network edges as possible. Admission control is also expected to run only on aggregate traffic and on a longer time scale.

3.4 The Tenet Scheme

In the Tenet scheme [34, 33] a framework for building real-time networks providing guaranteed performance is discussed. In this scheme the underlying packet switching network is assumed to be based on virtual circuits. An end to end connection is mapped to a virtual circuit. Quality of Service (QoS) guarantees are provided to individual connections.

The scheme provides deterministic as well as statistical guarantees. The QoS is represented by the the delay bound D, violation probability Z and loss probability p. According to the contract the probability that a packet exceeds the delay bound D is less than or equal to Z. The probability of a packet loss due to buffer overflows is bounded by p. This becomes deterministic guarantee if Z is equal to 0 and p is equal to 0. The traffic is specified by four parameters (X_{min}, X_{avg}, I, S_{max}) where X_{min} is the minimum inter-packet spacing of the traffic, X_{avg} is the worst case average inter-packet spacing over any interval of length I, and S_{max} is the maximum packet size which will be sent through the connection. Note that X_{min} gives the peak rate and X_{avg} gives the average rate of the traffic.

During the connection setup, the path from source node to the destination node is traversed and best possible resources are tentatively allocated at each switching node. During this forward phase of connection setup if it is found that the resources are inadequate to provide the requested QoS, the connection setup is stopped and all the allocated resources are deallocated. The application is informed about the failure of connection setup. However, if the network resources are adequate for the requested QoS, then after reaching the destination node the excess resources are divided equally among all the switching nodes in the path. The excess resources are deallocated and rest are committed on the way back from destination to source. The application is informed of the successful connection setup.

At any point in time the application may request to tear-down the connection. The virtual path of the connection is deleted and the resources allocated to the connection are deallocated. The application may be billed for the network usage.

A major criticism of deterministic guarantees is that the network has to reserve the resources for the worst possible case. This translates to peak rate reservation for bandwidth. Thus the network utilization is contemplated to be low. However, this is not true. In [35] it has been shown that for variable bit rate traffic peak rate reservation is not required.

We evaluate the performance of these admission control tests using actual real-time data. The data used consists of a variety of 1-2 hours long traces of VBR traffic generated by compressing real-time video using different coding algorithms. The characteristics of the video traces are discussed in detail in Chapter 2.

EDF scheduling algorithm, which is optimal is assumed for the purpose of analysis. Better admission control tests are derived for EDF scheduling algorithm. The tests are based on the generic burstiness function traffic descriptor described in [24].

The performance parameters we use are delay bound and long term network utilization[1]. We show how a proper choice of traffic model and admission control tests can help in improving delay bounds and network utilization. There is a clear

[1]Throughout this chapter, by utilization we mean the network utilization due to variable bit rate real-time channels requiring deterministic guarantees. There can be other real-time and non real-time traffic in the network at a lower priority level. Thus the total network utilization will be a lot better than the utilization we discuss in the chapter.

improvement in network performance by using leaky bucket traffic descriptor instead of Tenet group's X_{min}, X_{avg}, I traffic descriptor. This improvement is due to better admission control tests which make use of statistical multiplexing to give better network performance.

The rest of this chapter is organized as follows. We begin with properties of EDD scheduler in section 3.5. In section 3.6 admission control tests for EDD scheduler are derived. In section 3.7 the performance of the network is evaluated. A comparison of the two traffic descriptors is done from the point of view of network utilization. The chapter concludes in section 3.8.

3.5 Scheduling Algorithms

A number of scheduling algorithms have been proposed in the literature like rate controlled static priority [128], weighted fair queueing [86], hierarchical round robin [68] for providing QoS guarantees to the network traffic.

Our purpose is to evaluate the performance of deterministic guarantees. Therefore we do not attempt to compare different scheduling algorithms. Instead we use the the best scheduling algorithm for our purpose of performance evaluation.

Earliest Deadline First (EDF) scheduling algorithm is the optimal scheduling algorithm. Suppose there are N connections from which packets are scheduled on an output link. Let connection i have a local delay bound of d_i. Suppose packet j of connection i arrives at time a_{ij}. With EDF scheduling the packet is stamped with a deadline of $a_{ij} + d_i$ upon its arrival. The packets are scheduled on the output link in increasing order of their deadlines. The packet with earliest deadline is scheduled first.

We state without proof the well known fact that EDF scheduling algorithm is optimal if the traffic is known in advance.

Theorem 3.5.1 *Let there be a set of M jobs with deadline of job i equal to d_i. Also assume that the job i takes s_i of service time. If the given set of jobs can be scheduled for service such that no job misses its deadline, then EDF scheduler also schedules* *the jobs such that no job misses its deadline.*

This theorem can easily be extended to the on-line case when the timings of packet arrivals are not known in advance. Detailed discussion of the EDF scheduler can be found in [47].

Admission control test for providing deterministic guarantees using Delay-EDF scheduling discipline were first proposed in [123]. The admission control tests are based on the X_{min}, X_{avg}, I traffic model and are not optimal. These tests don't make full use of the traffic descriptor. Peak bandwidth is reserved during resource allocation which means that the parameters X_{avg} and I are not at all used during admission control.

We have developed improved admission control tests for EDF scheduler using generic traffic descriptor: the burstiness function $b(t)$. To simplify admission control tests, we approximate the burstiness function as a piecewise linear function. Unlike the previous tests the new admission control tests for Delay-EDF scheduling discipline do not reserve bandwidth at peak rate. Therefore the new tests are expected to improve the network utilization.

3.6 Admission Control for EDF scheduler

Let us assume a single connection with burstiness function $b(t)$. Recall from Chapter 2 that $b(t)$ bounds the maximum data sent in any interval of length t. Let us also assume without loss of generality that the backlog at time $T = 0$ is zero. Assume that a first packet in the connection arrives at time $T = 0$. Amount of data arriving in the interval $[0, T]$ is less than $b(T)$. Assume that the link is continuously busy in the interval $[0, T]$. So, the amount of traffic served by the link in the interval is lT, where l is the link rate. Therefore, backlog at time T is less than or equal to $b(T) - lT$. Since packets from a connection are served in first come first serve manner, a packet arriving when the backlog is maximum will experience maximum delay. The scheduling delay of a packet arriving at time T is equal to the time taken in clearing the backlog at time T, which is less than or equal to $\frac{1}{l}b(T) - T$. Hence the maximum scheduling delay is $\max_{T\geq 0}(\frac{1}{l}b(T)-T)$.

Now assume that there are N connections labelled from 1 to N. Let the delay bound of connection i be d_i and its bounding burstiness function be $b_i(.)$. Let a_{ik} be the arrival time of kth packet (p_{ik}) of connection i. Assume without loss of generality that busy period of scheduler starts at time 0. We limit our discussion to this busy period. We will first bound the amount of traffic which may be serviced before the packet p_{ik}. From this bound, we compute the time by when this packet is guaranteed to be serviced. Thus the delay bound for p_{ik} may be obtained. This gives the necessary and sufficient conditions for admission control.

The packet p_{ik} is tagged with a deadline of $a_{ik}+d_i$. Any traffic on connection i arriving before time a_{ik} will be serviced before the packet p_{ik} since it is tagged with a deadline less than $a_{ik}+d_i$. Amount this traffic (including the p_{ik}) is less than or equal to $b_i(a_{ik})$. Any packet of connection j arriving before time $a_{ik}+d_i-d_j$ will have a deadline less than $a_{ik}+d_i$, and hence may be serviced before the packet p_{ik}. Amount of such traffic of connection j is bounded by $b_j(a_{ik}+d_i-d_j)$. Therefore total such traffic is bounded by $\sum_{j=1}^{N} b_j(a_{ik}+d_i-d_k)$. It is also possible that a packet with a deadline greater than $a_{ik}+d_i$ is serviced before p_{ik}. To see this, assume that $d_j > d_i$ for some j, and a packet of connection j arrives at time 0 and immediately after this, at time $T=0^+$, a packet connection i arrives. Since at time 0, there was no other packet, the first packet of connection j is served. Deadline of this packet is d_j which is greater than d_i.

We first assume that no packet with deadline greater than $a_{ik}+d_i$ is serviced before p_{ik}. We will work with more generic case in next step. Now, the maximum traffic (of packets tagged with deadline less than or equal to $a_{ik}+d_i$) which has to be serviced before p_{ik} is $\sum_{j=1}^{N} b_j(a_{ik}+d_i-d_j)$. All of this traffic has to be serviced by time $a_{ik}+d_i$ to meet the delay bound for packet p_{ik}. Since the scheduler is never idle till p_{ik} is serviced, time when p_{ik} is serviced (s_{ik}) is given by:

$$s_{ik} = \frac{1}{l}\sum_{j=1}^{N} b_j(a_{ik}+d_i-d_j) \tag{3.1}$$

The delay of packet p_{ik}, is $s_{ik}-a_{ik}$ which should be less than d_i. Therefore $s_{ik}\leq$

$a_{ik} + d_i$. Thus we have the following sufficient condition:

$$\frac{1}{l}\sum_{j=1}^{N} b_j(a_{ik} + d_i - d_j) \quad \leq \quad (a_{ik} + d_i) \tag{3.2}$$

$$\Leftrightarrow \sum_{j=1}^{N} b_j(a_{ik} + d_i - d_j) \quad \leq \quad l(a_{ik} + d_i) \tag{3.3}$$

Since all packets of the connection must meet their delay bounds, it is sufficient to satisfy the following condition:

$$\forall_k \sum_{j=1}^{N} b_j(a_{ik} + d_i - d_j) \quad \leq \quad l(a_{ik} + d_i) \tag{3.4}$$

Substituting t in place of a_{ik} we get following sufficient condition for the admission control:

$$\forall_{t \geq 0} \sum_{j=1}^{N} b_j(t + d_i - d_j) \quad \leq \quad l(t + d_i) \tag{3.5}$$

$$\Leftrightarrow \forall_{t \geq d_i} \sum_{j=1}^{N} b_j(t - d_j) \quad \leq \quad lt \tag{3.6}$$

$$\Leftarrow \forall_{t \geq 0} \sum_{j=1}^{N} b_j(t - d_j) \quad \leq \quad lt \tag{3.7}$$

Now consider the case when one or more packets with deadline greater than $a_{ik}+d_i$ are serviced before p_{ik}. Let s_{ik} be the time at which p_{ik} is serviced. Let t' be the largest time less than s_{ik}, when a packet with deadline greater than $a_{ik} + d_i$ begins service. Since t' is largest such instant, all the packets served in the interval $(t', s_{ik}]$ have deadline less than or equal to $a_{ik} + d_i$. Also, $t' \leq a_{ik}$ or else the scheduler will schedule p_{ik} instead of the packet with greater deadline. This implies that the delay bound of packet serviced at time t' is larger than d_i. Moreover, all the packets backlogged at time t' (including the packet serviced at time t') have a deadline larger than $a_{ik} + d_i$. Therefore none of the packets backlogged at time t' will be serviced in the interval $(t', s_{ik}]$. Thus, except for the packet serviced at time t', only the packets arriving in the interval $[t', s_{ik}]$ may be served in the interval. Out of these packets, only the packets having a deadline less than $a_{ik} + d_i$ will be serviced before p_{ik}. The amount of traffic of connection j, arriving after time t' with deadline less than $a_{ik}+d_i$

is bounded by $b_j(a_{ik} + d_i - t' - d_j)$. Therefore the maximum traffic serviced at or after time t' till p_{ik} is serviced is bounded by $\sum_{j=1}^{N} b_j(a_{ik} + d_i - d_j - t') + S$, where S is the size of packet which began service at time t'. In order to meet the delay bound of the packet p_{ik}, all of this traffic must be serviced by time $a_{ik} + d_i$. Therefore it is sufficient to satisfy the following equation to satisfy the delay bound condition for packet p_{ik}:

$$\frac{1}{l}(\sum_{j=1}^{N} b_j(a_{ik} - t' + d_i - d_j) + S) \leq a_{ik} - t' + d_i \tag{3.8}$$

$$\Leftrightarrow \sum_{j=1}^{N} b_j(a_{ik} - t' + d_i - d_j) + S \leq l(a_{ik} - t' + d_i) \tag{3.9}$$

Since all packets of the connection must meet their delay bounds, it is sufficient to have:

$$\forall_k \sum_{j=1}^{N} b_j(a_{ik} - t' + d_i - d_j) + S \leq l(a_{ik} - t' + d_i) \tag{3.10}$$

Substituting t in place of $a_{ik} - t'$ we get following sufficient condition for the admission control:

$$\forall_{t \geq 0} \sum_{j=1}^{N} b_j(t + d_i - d_j) + S \leq l(t + d_i) \tag{3.11}$$

$$\Leftrightarrow \forall_{t \geq d_i} \sum_{j=1}^{N} b_j(t - d_j) + S \leq lt \tag{3.12}$$

If d_{min} is the minimum delay bound among all the connections, and S_{max} is the maximum packet size, then we have:

$$\forall_{t \geq d_{min}} \sum_{j=1}^{N} b_j(t - d_j) + S_{max} \leq lt \tag{3.13}$$

$$\Leftrightarrow \max_{t \geq d_{min}} (\sum_{j=1}^{N} b_j(t - d_j) + S_{max} - lt) \leq 0 \tag{3.14}$$

Note that the above expression doesn't depend on i. This implies that the test for all the connections is the same. So the test is needed only once when a new connection is added. If the test is satisfied then packets from all the connections will meet their delay bound.

3.6.1 Computation of admission control tests

Note that the burstiness function of a traffic is based on the worst case traffic pattern of the connection. It may be obtained from the traffic trace and is typically not a mathematically well defined function. Therefore, it is difficult to maximize the left hand side of Equation 3.14 in general. The burstiness function can be approximated as a piecewise linear function, with sufficient number of segments. In this case, the function to minimize on the left hand side of Equation 3.14 also becomes piecewise linear. A piecewise linear function is optimized either at boundaries or at its points of discontinuity of slope. Therefore the required maxima may be computed by evaluating the function at all the segment boundaries and then taking the maximum of the computed values. Note that the test will be satisfied at $t = \infty$ if and only if $\lim_{t\to\infty} \sum_{j=1}^{N} \frac{1}{t} b_j(t) \leq l$. This is same as saying that the sum of the average bit rate required by all the connections is less than the link rate.

If the number of segments is large, then computation of this test becomes inefficient. In most of the traffic models, the number of segments is small. The leaky bucket model represents the traffic using two parameters: The burst size σ and the service rate ρ. The amount of traffic sent in any interval of length t is bounded by $\sigma + \rho t$. The burstiness function can represent this model accurately. The burstiness function for the leaky bucket model is given by $b(t) = \sigma + \rho t$ for $t \geq 0$ and $b(t) = 0$ for $t < 0$. Note that this function is piece wise linear with two segments. In order to compute the admission control test, we only need to evaluate the expression $f(t) = (\sum_{j=1}^{N} b_j(t - d_j) + S_{max} - lt)$ at $t = d_i$ for all i. To carry out this computation, we arrange the connections in increasing order of their delay bounds. Let the indices of the connection be rearranged such that $d_1 \leq d_2 \leq d_3 \ldots$. We start by evaluating $f(t)$ at d_1 and from this value we evaluate $f(d_2)$ and then $f(d_3)$ and so on. For leaky bucket model, the burstiness function of connection j is given by $b_j(t) = \sigma_j + \rho_j t$. So,

$$f(d_1) = \sigma_1 + S_{max} - ld_1 \tag{3.15}$$

We also maintain $g_i = \sum_{j=1}^{i} \rho_j$. Now $f(d_i)$ and g_i are related to $f(d_{i-1})$ and g_{i-1} by

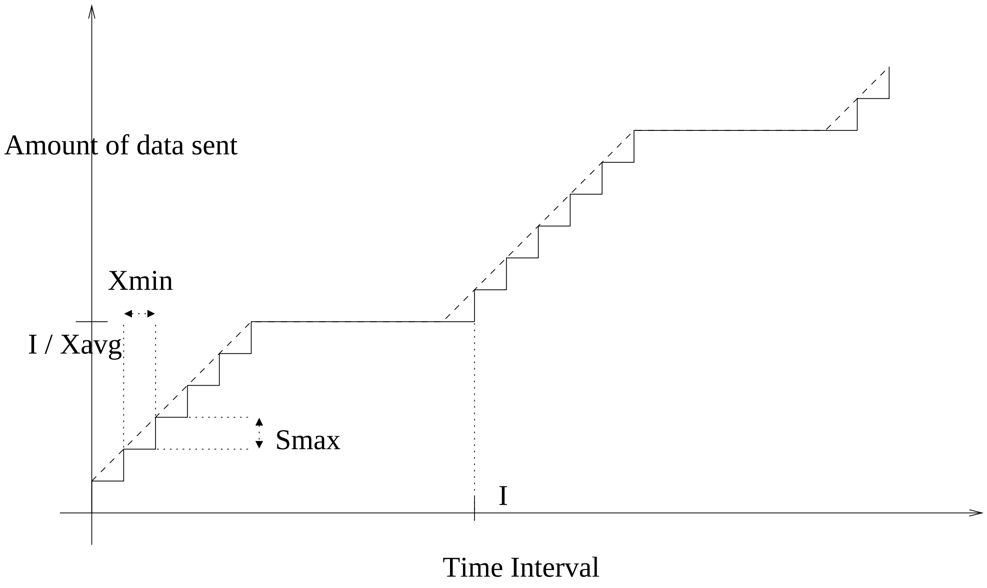

Figure 3.1: Burstiness function for Tenet traffic model.

the following expressions:

$$f(d_i) = f(d_{i-1}) + (g_{i-1} - l)(d_i - d_{i-1}) + \sigma_i \tag{3.16}$$

$$g_i = g_{i-1} + \rho_i \tag{3.17}$$

Using the above expressions, the admission control tests can be computed in $O(N)$ time for the leaky bucket traffic model, where N is the number of connections.

In the Tenet scheme [34, 33], the traffic is described using four parameters, $X_{min}, X_{avg}, I, S_{max}$. X_{min} is the minimum inter-packet spacing of traffic generated by the source. X_{avg} is the average inter-packet spacing of traffic generated by the source. I is the interval over which X_{avg} is measured. The minimum value of X_{avg} is taken for all the intervals of length I. This gives the worst case average rate over all intervals of length I. S_{max} is the maximum packet size sent by the source. The burstiness function for this model is given by:

$$b(t) = \left(\min\left(\left\lceil \frac{t \bmod I}{X_{min}} \right\rceil, \left\lceil \frac{I}{X_{avg}} \right\rceil\right) + \left\lceil \frac{t}{I} \right\rceil * \left\lceil \frac{I}{X_{avg}} \right\rceil\right) * S_{max} \tag{3.18}$$

This is a step function with steps at $mI + nX_{min}$ where m, n are integers and $m \geq 0$ and $\frac{X_{avg}}{X_{min}} \geq n \geq 0$. Intuitively, in the beginning the source may send packets

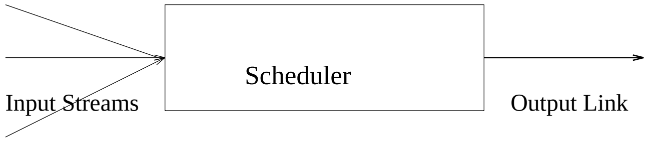

Figure 3.2: Network model.

at peak rate (with inter-packet spacing of X_{min}) but it may only send $\frac{I}{X_{avg}}$ packets. After this it has wait till time I to meet the average rate constraint. At time I the source may again send a burst of packets at the peak rate. This is shown in Figure 3.1. To reduce the number of segments in the burstiness function, we replace the steps due discrete packets sent at a spacing of X_{min} by a bounding straight line. We get a bounding burstiness function as shown in Figure 3.1. This bounding burstiness function is given by:

$$b(t) = \left(\min\left(\frac{(t + X_{min}) \bmod I}{X_{min}}, \left\lceil \frac{I}{X_{avg}} \right\rceil\right) + \left\lceil \frac{t}{I} \right\rceil * \left\lceil \frac{I}{X_{avg}} \right\rceil\right) * S_{max} \tag{3.19}$$

(3.20)

Typically, the value of I is significantly greater than the delay bounds d_js. So, the maxima in Equation 3.14 is usually at points when t is either d_j or $I_j \frac{X_{minj}}{X_{avgj}}, I_j$.

3.7 Performance Evaluation

In this section we evaluate the admission control tests for two models: X_{min}, X_{avg}, I model and σ, ρ model. The main focus of our study is to find out the best achievable network utilization for variable bit-rate real-time channels while providing deterministic guarantees. We also study which of the two models describing traffic characteristics is better suited for providing deterministic guarantees. The parameters we use to evaluate the models are best possible delay bound which can be promised by the network and the long term network utilization. We also investigate if it is possible to improve the network utilization while still providing deterministic guarantees.

The analysis assumes a single node with bounded buffer as shown in figure 3.2.

The parameters of the switch were taken from XUNET II switch [40]. The outgoing link was assumed to have bandwidth of 200 megabits per seconds (MBPS) and the total buffer space was assumed to be 32 M Bytes.

One could possible carry out analysis of deterministic guarantees in the context of a more general network setting having multiple nodes. In case of a general network other parameters like network topology, distribution of connections, bottlenecks in network etc. come in the picture. In order to keep the number of variable parameters as small as possible we focus on a single node analysis.

The admission control tests were evaluated for homogeneous traffic types with each connection having same QoS requirements. The QoS parameter of main concern is the delay bound. The performance parameters are the maximum number of homogeneous network connection with a specified delay bound, the scheduling region and the network utilization.

The delays we consider are the end to end delays as perceived by the application. For example an application sending video in form of JPEG frames can deliver one frame full at a time to the network at the intervals of 30 frames per second. The network will then break the frame up into packets of smaller size to be able to send them. The network may decide to space these small packets uniformly over a frame interval (as done by RTIP). In this case the maximum end to end delay as perceived by the application will be the sum of maximum packet delay of network and one frame time. We assume that for X_{avg}, I model the network breaks up the frame into packets of size 2K bytes and spaces them uniformly over one frame interval. Since we don't have any control over transmission and propagation delays we exclude them.

For a given trace, graphs of X_{avg} vs I and σ vs. ρ were constructed. It was assumed that the traffic parameters of all the connections are same as those of the trace being considered (i.e. the traffic is homogeneous with same QOS requirements). For example if trace NEWS was being considered then it was assumed that all the connections have traffic characteristics same as that of the trace NEWS. A graph between the minimum achievable delay and number of connections accepted was then plotted by choosing the best possible value of $(\sigma,\ \rho)$ or $(X_{avg},\ I)$ from the σ vs. ρ or X_{avg} vs I graphs constructed. The best possible delay bound is computed by varying the parameters of the of the model to all possible values and choosing the ones which give the smallest delay bound. For example in case of σ, ρ model the range of meaningful values which (σ and ρ) can take is given by set of all points in the graphs of figure 2.7, 2.11 and 2.14. Given number on connections, all possible values of (σ, ρ) were tried to get a delay bound. The value of (σ, ρ) which gives the least bound was selected. It turns out from the nature of the σ-ρ graphs that one doesn't have to check for all possible values of (σ, ρ). The largest value of ρ using which the bandwidth test is satisfied gives the best possible delay bound. Similar analysis was carried for the tenet model also. Graphs are drawn for each of the 10 traces being considered.

Figures 3.3, 3.4 and 3.5 show the results. On X-axis is the number of homogeneous connections set up. Y-axis gives the best possible delay bound for those set of connections. The number of connections vary from 1 to the point when no more connections can be accepted for all possible parameters describing the traffic. Figure 3.3 shows the results for traffic generated by JPEG compressed video, figure 3.4 shows it for MPEG compressed video and figure 3.5 for NV type of traffic. For a comparison of the two models, graphs for both the models are plotted for them in the same figure. The scheduling region for any trace is the area above and left of the its corresponding curve in figures 3.3, 3.4 and 3.5.

It is apparent that using σ, ρ model gives a better delay bound in all the cases. Also in all the cases by using σ, ρ model network is able to accept more connections as compared to using X_{avg}, I model.

3.7.1 Performance on JPEG Compressed Video

From figure 3.3 one can observe that for JPEG compressed video there is knee in the curve for both the models after which as more connections are accepted the delay bound shoots up rapidly. This knee corresponds to the point when the sum of peak rates of all the connections become equal to the link bandwidth and hence the position of the knee is same for both the models. The number of additional connections which can be accepted after the knee is small. This essentially means that queuing inside

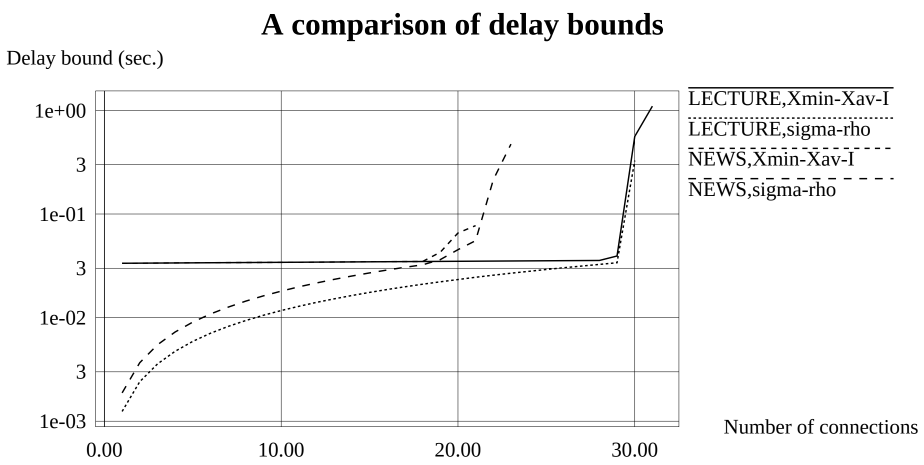

Figure 3.3: A comparison of delay bounds for JPEG compressed video.

the network can't smooth out the burstiness of the video traffic generated by similar JPEG encoders and one has to reserve bandwidth close to peak in order to provide deterministic guarantees.

The delay bound when the number of connections is less than the knee point is fairly constant for X_{min}, X_{avg}, I model and is close to one frame time. This delay is incurred at the point when the frames are broken up into individual packets. The queueing delay in the network is negligible as compared to this delay. For the σ, ρ model we assume that there is no peak rate control. The frame is broken up into packets which are sent back-to-back on the network. The delay bound starts with a small value and eventually catches up with that of the tenet model. However this doesn't imply that peak rate control is undesirable. We should take into account that above is just a single node analysis. If the packets are sent through multiple nodes then in case of the X_{min}, X_{avg}, I model the delay due to peak rate control is incurred only once. The queueing delays per node are added up. Since the queueing delays are small the total increase in delay bound for a multiple hop path remains small. For σ, ρ model there is no delay due to peak rate control. The delay is only the queueing delay. When the same analysis is extended to multiple nodes the total delay bound is expected to increase linearly with number of nodes in the path. Peak rate control and small packet size seems necessary for providing low delays in large networks. If peak rate control is to be done then this model needs to be relaxed to a model with 4 parameters (σ_1, ρ_1, σ_2, ρ_2) where σ_1, ρ_1 represent maximum packet size of the network and peak rate and σ_2, ρ_2 represent the average rate. The generalized admission control tests developed can easily be used for this model.

3.7.2 Performance on MPEG compressed video

Figure 3.4 shows that delay bounds and number of accepted connections are better for the σ, ρ model. We should be careful in concluding anything about delay bounds. The delay bounds for this model may increase much faster than that for the X_{min}, X_{avg}, I as the number of nodes in the path increases. However with the proposed modification of the model and peak rate control, both the models are expected to give similar results even for large networks. However it is very clear that using the σ, ρ model enables the network to accept more connections. This is expected to remain so in large network also. Another noticeable observation is the knees in the graphs. The delay bound increases rapidly when additional connections are accepted beyond the knee. The number of additional connections accepted after the knee is small. This behavior resembles the behavior on JPEG encoded video. The position of knee corresponds to the position of knee in the σ, ρ curves. This gives us a simple and powerful method to select appropriate parameters for the σ, ρ model. The optimal (σ, ρ) remain close to the knee for most of the time. Whereas I of the X_{min}, X_{avg}, I model increases smoothly as the number on connections is increased. Thus there is a problem in choosing a value of I. If I is chosen to be small then the connection will reserve more resources than actually needed and would result in worse utilization of the network. On the other hand if a large value of I is chosen then the network will not be able to guarantee the best possible delay bound. The guaranteed delay bound will be higher than the one which can be obtained using the σ, ρ model.

3.7.3 Performance on NV type traffic

In figure 3.5 comparison of the two models for NV type traffic is done. The σ, ρ model outperforms the X_{min}, X_{avg}, I model both in terms of delay bounds and number of

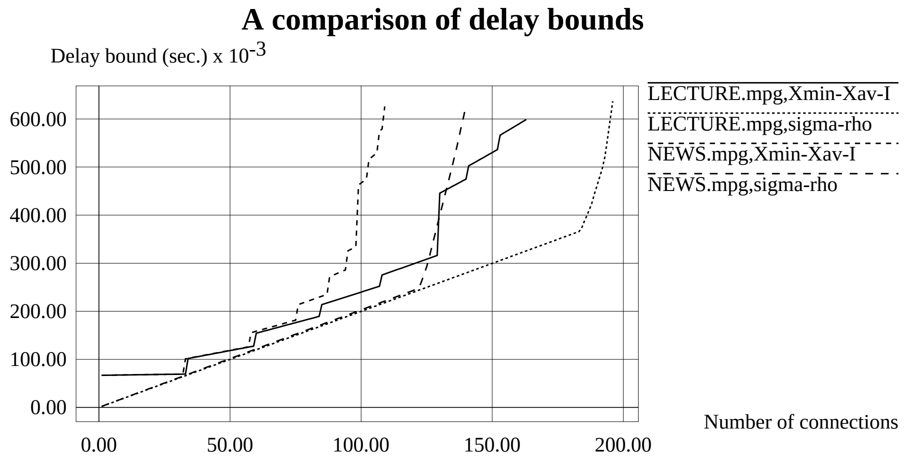

Figure 3.4: A comparison of delay bounds for MPEG compressed video.

connections accepted. It is interesting to note that although NV0 and NV2 are traces of two different video streams, the graphs for both of them are extremely close to each other. In the trace NV1 the rate control of the NV software was probably made higher for about 3 minutes which results in a different behavior. Thus in the absence of any change in parameters of the coding algorithm for the NV software, the characteristics of the traffic generated is nearly independent of the contents of the video being sent.

Another noticeable observation is that the delay bounds given by the σ, ρ model are significantly better as compared to the ones given by the X_{min}, X_{avg}, I model. Since the data was already packetized no additional peak rate control was done for any of the models. This observation reinforces that the σ, ρ model is able to characterize the traffic in a way which is closer to the actual traffic. As a results tighter bounds are achieved.

3.7.4 Analysis of Network Utilization

Table 3.1 presents a comparison of maximum achievable network utilization of the network using the two traffic models. For a particular trace all the connections are assumed to have characteristics same as that of the trace. Then maximum number of homogeneous connections that can be set up through the single node is computed. The delay is set to unlimited so that delay is never a bottleneck. Utilization tells

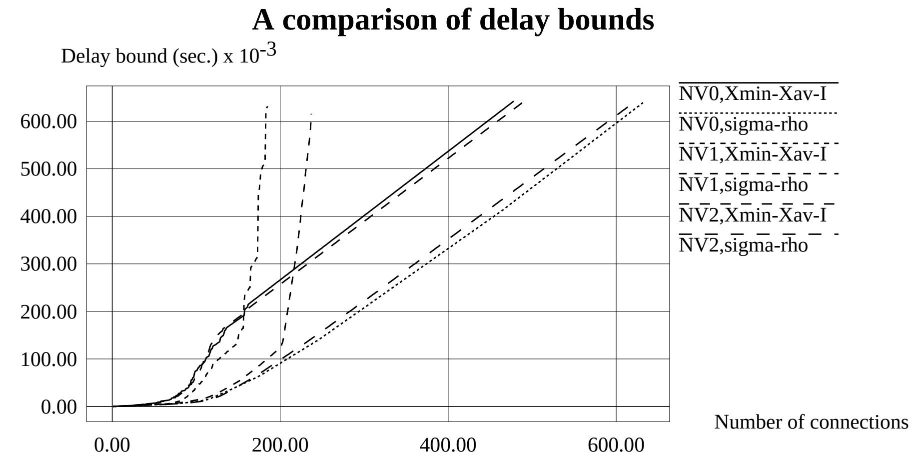

Figure 3.5: A comparison of delay bounds for traffic type produced by NV.

the maximum achievable network utilization through a single node and traffic having same characteristics and same QOS requirements. Multiplexing gain is the ratio of sum of peak rates to the link bandwidth.

For JPEG compressed video the utilization of the network varies from 0.334 to 0.700. A part of this variation in utilization is due to the fact that different traces are being used. Another part is because of different traffic descriptors and hence different admission control tests.

For example the variation in network utilization for the trace NEWS is from 0.454 to 0.580 is just because of using a better traffic descriptor. The σ, ρ model performs better as compared to the other model. The multiplexing gain varies from 0.940 to 1.411. Although the admission control tests for Delay-EDF scheduling disciplines using the X_{min}, X_{avg}, I model reserve bandwidth at peak rate, their performance is not terribly worse.

For MPEG compressed video the X_{min}, X_{avg}, I model for Delay-EDF scheduling discipline performs significantly worse than all the other cases. This is so only because the current admission control tests for Delay-EDF scheduling discipline are not tight enough for the Tenet traffic descriptor. The current tests carry out reservation of bandwidth at peak rate which is not required. The generalized admission control tests developed can be adopted for both the models. The σ, ρ model gives a better

	X_{min}, (X_{avg}, I)	model	(σ, ρ)	model
Trace	Utilization	Multiplexing gain	Utilization	Multiplexing gain
NEWS	0.454	0.966	0.580	1.234
MUSIC	0.402	0.968	0.592	1.427
LECTURE	0.563	0.974	0.603	0.941
SPORT	0.639	0.998	0.700	1.092
MOVIE	0.334	0.940	0.500	1.411
LECTURE.mpg	0.101	0.994	0.598	5.906
NEWS.mpg	0.084	0.993	0.357	4.227
NV0	0.023	0.994	0.401	17.449
NV1	0.021	0.978	0.143	6.817
NV2	0.021	0.998	0.314	15.122

Table 3.1: Network utilization for deterministic real-time channels.

network utilization in all the cases. The multiplexing gain goes as high as 5.906 which means the the required bandwidth to send a MPEG compressed video can be as low as one fifth of the sum of the peak rates. Peak rate reservation which has been one of the major criticism of deterministic guarantees is not at all needed if one uses proper traffic model and admission control tests. The best achievable network utilization can be up to 0.598 (for real-time VBR only traffic) which is very encouraging.

Traffic generated by the NV software is a lot more burstier than other traffic types discussed. Thus the network utilization is smaller and the multiplexing gain is larger. If we exclude the Delay-EDF scheduling discipline with X_{min}, X_{avg}, I model then the network utilization varies from 0.112 to 0.401 and the multiplexing gain varies from 6.817 to 17.449.

3.8 Conclusions

Peak rate reservation is not necessary for providing deterministic guarantees. The multiplexing gain reaches up to 17 in our experiments. Use of appropriate traffic models and admission control tests is necessary to achieve such a high utilization. The σ, ρ model is shown to produce better results in all the cases. The generalized admission control tests are developed which can be specialized for multiple leaky bucket descriptor model. We expect that two leaky bucket descriptor will outperform single leaky bucket traffic descriptor. However this has to be substantiated with proper studies.

The analysis done above is a single node analysis of network carrying homogeneous traffic. Thus the utilization figures obtained are in some sense a upper bound on maximum achievable utilization. Heterogeneous traffic having different QoS requirements is expected to put more stringent demands on the network. Thus the average utilization is expected to be lower in a more general setting. A more extensive study is needed to see the effect of heterogeneous traffic with different QoS requirements.

Another limitation of the current study is lack of study on network with multiple nodes Our study has given a good insight into the network performance. One need not consider Tenet traffic descriptor anymore while carrying out multiple node analysis. In a network having multiple nodes various other factors like resource fragmentation, bottlenecks etc. come in play and reduce the overall network utilization. It would be very interesting to study these factors.

The results obtained are fairly general and consistent. For three totally different type of traffic the results seem to agree. Therefore with a high confidence we can say that these results are going to remain valid even for other types of traffic.

Chapter 4

RRR: Recursive Round Robin Scheduler

Scheduling has been an interesting problem since its inception. In the context of real-time networks, a scheduling algorithm is concerned with dispatching streams of packets sharing the same bandwidth such that certain guaranteed performance for each stream like rate and delay bound is provided. Packet scheduling is required in many network elements such as host adaptors, routers and switches. In this chapter, we propose and discuss a new scheduling algorithm which we call *recursive round robin* (RRR) scheduler for scheduling fixed size packets in an integrated services network. It is based on the concept of a scheduling tree in which distinct cell streams are scheduled recursively. Special emphasis is placed on the design and analysis of the scheduler. A delay bound is analytically derived for the scheduler and verified using simulation. The scheduler can work in either a work-conserving mode or non-work-conserving mode. It is shown that the work-conserving version of the scheduler is fair. Fairness indices for the work-conserving scheduler are analytically derived. The simple nature of this algorithm makes it possible to implement it at very high speeds, while considerably reducing the implementation cost. Another variant of the scheduler can schedule variable sized packets. The delay and fairness properties of the variable sized packets scheduler are looser than the fixed sized packet scheduler, and its time complexity is little higher.

4.1 Introduction

One of the most important issue in providing QoS in an integrated service network is the choice of packet scheduling algorithms at the switches. In a packet switching network, packets from different connections compete for some shared resources at each switch. Without proper control, these interactions may adversely affect the network performance experienced by clients. The scheduling algorithm at the switching nodes, which controls the relative ordering in which packets from different connections are serviced, determines how packets from different connections interact with each other.

Although scheduling algorithms and associated performance problems have been widely studied in the context of hard real-time systems and queueing systems, results from these studies are not directly applicable in the context of providing guaranteed QoS to connections in an integrated services network. Analyses of hard real-time systems usually assume a single server environment, periodic jobs, and the job delay is bounded by its period [114]. However, the network traffic is bursty, and the delay constraint for each individual connection is independent of its bandwidth requirement. In addition, bounds on end to end performance need to be guaranteed in a networking environment, where traffic dynamics are far more complex than in single server environment. Queueing analysis is often intractable for realistic traffic models. Also, classical queueing analysis concentrate on average performance for aggregate traffic [74, 75, 126], while in an integrated services network, performance bounds need to be derived on a per connection basis [31, 76, 5]. In addition to the challenge of providing end to end, per connection performance guarantees to heterogeneous and bursty traffic, scheduling algorithms must be simple so that they can be implemented at very high speeds.

In general, schedulers can be characterized as work-conserving or non-work-conserving. A scheduler is said to be work-conserving if the scheduler is never idle when at least one packet is buffered in the system. A non-work-conserving scheduler may remain idle even if there are available packets to transmit. Recently, a number of new scheduling algorithms that are aimed to provide per connection QoS guarantees have been proposed in the context of integrated services networks [47, 130, 128, 120, 26, 86, 100, 131, 8, 9, 129, 52, 115, 116, 119, 122, 68, 51, 99].

Many of these scheduling algorithm, were designed for scheduling packets of variable sizes and were complex [26, 131, 128]. Some of them were later adopted for scheduling ATM cells [122]. However the inherent complexity remains and typical operations needed to schedule a cell are addition, multiplication and division. In addition they also need a multi-level priority queue. Among the schedulers optimized for scheduling ATM cells, some have poor delay and fairness properties [87], some are not scalable for fine rate granularity [99] and some may need to over-allocate rate resulting in poor utilization of link bandwidth [68].

In this chapter we propose a new scheduling algorithm called *recursive round robin scheduler* (RRR), which is optimized for scheduling fixed size packets. The proposed scheduler is based on the concept of a scheduling tree in which distinct streams are scheduled recursively. In order to schedule a cell, the scheduling tree is traversed starting from its root down to a leaf. While traversing the tree, the data stored in its nodes is modified using simple addition or bit manipulation operations. The scheduling tree is modified each time a stream is added or deleted at the link. The rate of each packet stream is represented in a floating point binary fraction normalized with respect to the link capacity. Most of the properties of the scheduler depend upon the count c of number of ones in the binary representation of normalized rate. For a compliant stream of ATM cells of rate r, and bucket size σ, the scheduler provides a delay and jitter bound of $\frac{1}{r}(\sigma + c)$.

We show that the work-conserving version of the RRR algorithm is fair. The residual link capacity is evenly divided among active streams. The service fairness index of two streams i, j is $(\frac{c_i}{r_i} + \frac{c_j}{r_j})$ where r_i and r_j are rates of the two streams, c_i and c_j are the counts of number of ones in the binary representation r_i and r_j respectively. The worst case fair index of work-conserving RRR is bounded by $\frac{c}{r}$.

The algorithm has good scaling properties. The size of the scheduling tree is bounded by the number of streams times the rate granularity. In a naive software implementation, the time taken to schedule a packet, is proportional to the granularity of rate representation. The optimized algorithm for fixed packet size needs only bit manipulation operations to schedule a cell. Maximum time taken to add new streams or remove old streams is proportional to the maximum depth of the scheduling tree, which is equal to the log of ratio of link rate to minimum supported rate. However, by combining several nodes of the tree, the time complexity of software implementation can be further improved. As a result it can be implemented on high speed low cost ATM host adaptors where the scheduling overhead of the algorithm can be as little as two memory accesses to schedule an ATM cell. The hardware implementation of the fixed cell size RRR can operate at clock speeds. Its implementation cost varies linearly with the number of streams and the rate granularity. The algorithm is also suited for scheduling in high speed ATM switches, especially on the edge of ATM network where traffic policing and isolation is required.

We also generalize the RRR scheduling algorithm for scheduling variable sized packets. The delay and fairness properties of this variant are a little worse than that of the fixed size packet scheduler. Important concepts such as link sharing [38], class based queuing etc. can be implemented using the RRR scheduler.

This chapter is organized as follows. In Section 4.2 we discuss the properties of a good scheduler. In Section 4.3 description of the RRR scheduling algorithm is presented. Section 4.4 describes delay and fairness properties of the scheduler. Section 4.5 discusses variants derived out of RRR. Section 4.6 describes simulation results. Section 4.7 discusses implementation issues and tradeoffs for hardware and software implementations of RRR. The chapter concludes in Section 4.8.

4.2 Background

4.2.1 Model

We consider a network with arbitrary topology of links and switches. Packets arriving on an input link of a switch are switched to an output link depending upon their headers and forwarding tables at the switch. This switching is typically done by a cross-connect which is usually made fast to avoid excessive queueing at the input links [70]. Therefore, most of the packets are queued for transmission at the output links. With these assumptions, a connection in such a network can be modeled as traversing a number of queueing servers, with each server modeling the output link of a switch.

4.2.2 Quality of Service Parameters

The most important clause in the service model is the specifications of performance requirements and traffic characteristics. In the deterministic service model, the single most important performance parameter is the end-to-end delay bound, which is essential for many applications which have stringent real-time requirements. While throughput guarantee is also important, it is provided automatically with the amount specified by the traffic characterization. Another important parameter is the end to end delay jitter bound. The delay jitter for a packet stream is defined to be the maximum difference between delays experienced by any two packets [31, 32]. For continuous media playback applications, the ideal case would be that the network introduces only constant delay, or zero delay jitter. Having a bounded delay jitter service from the network makes it possible for the destination to calculate the amount of buffer space needed to eliminate the jitter. The smaller the jitter bound, the less the amount of buffer space needed. Since it is more important to provide end to end delay and delay jitter bounds than average low delays, packets arriving too early may not even be desirable in such an environment. In fact, the earlier a packet arrives , the longer it needs to occupy the buffer. This is one of the most important differences between the performance requirements of the bounded delay service and the best effort service provided by the traditional computer networks. Performance bounds are more important for guaranteed service while average performance indices are more important for the best-effort service.

A third important parameter is the loss probability. Packet loss can occur due to buffer overflow or delay bound violation. A statistical service allows non zero loss probability while a deterministic service guarantees zero loss. With a deterministic service, all packets meet their performance requirements even in the worst case. With a statistical service, stochastic or probabilistic bounds are provided instead of worst case bounds. Statistical service allows the network to over-book resources beyond the worst case requirements, thus one may increase the overall network utilization by exploiting statistical multiplexing gain.

4.2.3 Properties of Scheduling Algorithms

Together with traffic modeling and connection admission control algorithms, schedulers are the most important components for providing QoS guarantees. While connection admission control algorithms reserve resources during connection establishment time, packet scheduling algorithms allocate resources according to the reservation during data transfer. Three types of resources are being allocated by a scheduling algorithm: bandwidth, promptness and buffer space, which in turn affects three QoS parameters: throughput, delay and loss rate.

Although it is possible to build real-time services on top of a vast class of scheduling algorithms, it is desirable for the scheduling algorithm to satisfy certain properties.

Efficiency

To guarantee certain performance requirements, we need a connection admission control policy to limit the real-time traffic load in the network, or limit the number of real-time connections that can be accepted. A scheduling algorithm is more efficient than another one if it can meet the same end-to-end performance guarantees under a heavier load of guaranteed service traffic. An efficient scheduler will result in a higher utilization of the network.

Isolation

The network should protect well behaving real-time streams from three sources of variability: ill-behaving users, network load fluctuations and best-effort traffic. It has been observed in operational networks that ill-behaving users and malfunctioning equipments may send packets to a switch at a rate higher than declared. Also, network load fluctuations may cause a higher instantaneous arrival rate from a connection at a switch, even though the connection satisfies the traffic constraint at the entrance to the network. Another source of variability is the best-effort traffic. Although the real-time traffic load is limited by connection admission control, best-effort packets are not constrained. It is essential that the scheduling algorithm should meet the performance requirements of packets from well behaving real-time clients, even in the presence of ill behaving users, network load fluctuations and unconstrained best-effort traffic.

Sharing

Under-utilized links may have some spare bandwidth available. Even heavily loaded links may have spare bandwidth available for short intervals of time due to the variability in load. The scheduler should allow sharing of the residual bandwidth among connections in a fair manner. The distribution of excess bandwidth to connections should be done in proportion to their allocated rate. Thus, a connection with higher rate allocation (which is possibly paying higher charges) gets a larger share of the spare bandwidth. According to the generalized processor sharing (GPS) [100, 101] criteria, if two streams i and j are continuously backlogged in the interval $[t_1, t_2]$, then the amount of service received by them should be proportional to their allocated rates. If $W_i(t_1, t_2)$ represent the amount of service given to the stream i in the interval $[t_1, t_2]$ and r_i represent the rate allocated to stream i then:

$$\frac{W_i(t_1, t_2)}{r_i} = \frac{W_j(t_1, t_2)}{r_j}$$

GPS is an idealized service discipline based on the fluid flow model. It cannot be realized in discrete packet switching networks. Therefore, it is impossible to achieve perfect fairness in discrete packet scheduler. A measure of fairness based on the GPS criteria called service fairness index (SFI) has been proposed in [52] to compare fairness properties of discrete packet schedulers.

SFI is defined as the maximum difference between the relative services offered to two sessions which are continuously backlogged in the same interval. If sessions i and j are continuously backlogged in the interval $[t_1, t_2]$ then:

$$\text{SFI} \leq \left| \frac{W_i(t_1, t_2)}{r_i} - \frac{W_j(t_1, t_2)}{r_j} \right|$$

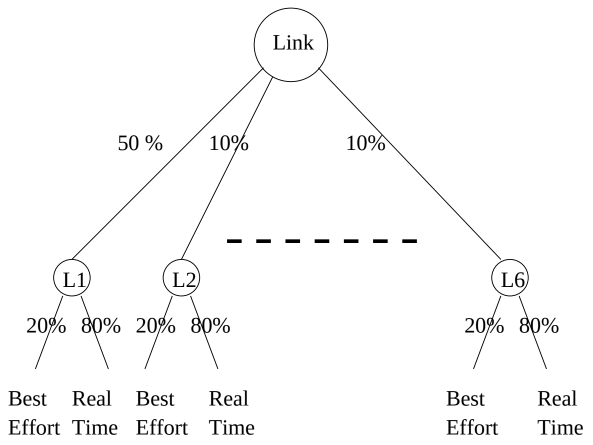

Figure 4.1: Example of a link sharing hierarchy.

Hierarchical Scheduling

The concept of link sharing has been proposed in [38]. Link sharing allows a link to be hierarchically split into several classes. Consider the example shown in Figure 4.1. The link is partitioned into six logical links, with the first link taking fifty percent of the link bandwidth and remaining five links taking ten percent each. These links may be assigned to six different organizations. In addition, to avoid starvation of best-effort traffic, twenty percent of the fifty percent assigned to the first link, is allocated to the best-effort traffic.

The hypothetical hierarchical GPS (H-GPS) sharing algorithm [129] defines a flexible framework to support hierarchical link sharing and traffic management for different service classes. Similar to GPS, the H-GPS is an idealized service discipline based on the fluid flow model. However the hierarchical versions of packetized approximations of GPS such as PGPS [100] do not work well to approximate H-GPS [9]. Packet scheduling algorithms to approximate H-GPS have been proposed in [8, 129]. The term worst case fairness index (WFI) has been defined [129] to evaluate how closely a scheduler can approximate H-GPS. A packet scheduler with small SFI and WFI may be stacked hierarchically to approximate H-GPS and thus implement the link sharing hierarchy. WFI is defined as follows:

The WFI for a stream i of rate r_i is less than or equal to $\frac{C_i}{r_i}$ if in any interval $[t_1, t_2]$, over which the stream is continuously backlogged, the amount of traffic of stream i scheduled is bounded by:

$$W_i(t_1, t_2) \geq (t_2 - t_1)r_i - C_i$$

Simplicity

The scheduling algorithm should be conceptually simple to allow tractable analysis and mechanically simple to allow high speed implementation.

The algorithms like WFQ or PGPS [86, 100], WF2Q [8], SCFQ [52] and EDF [47] tag each incoming packet with a stamp. At each step, a packet with smallest stamp is selected for scheduling. These algorithms need to maintain a sorted priority queue of the packets in order to select packets with the smallest stamp. If N streams are passing through a link, the inherent time complexity of each scheduling operation is at least $\log N$. Some algorithms like PGPS [100] and WF2Q [8] need to carry out a simulation of fluid flow system, which in the worst case, may take time proportional to N to schedule a packet.

The time available to make a scheduling decision is very small. For a 622Mbps ATM interface, the time available to carry out switching, copying and scheduling of cells is only 680ns. Therefore it should be possible to implement the scheduling algorithms in the hardware working at clock speeds, especially in the backbone switches. On a network interface card, a software implementation may be feasible but it can only make 4-5 memory accesses (memory accesses being the slowest operation) in order to make a scheduling decision.

4.3 Scheduler Description

We use the term stream to refer to flow or session. The term cell is used to describe packets of fixed size. Subscript i is usually used to identify a stream. For each stream (i) a rate is allocated at an output link. We refer to this rate as allocated rate (r_i).

We divide all the rates by the link rate. As a result the link rate becomes equal to 1. We refer to the rate so obtained as normalized rate. Further we will assume that the rate is represented in fixed point binary fraction with g bits of granular-

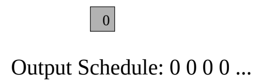

Figure 4.2: Scheduling tree (no stream).

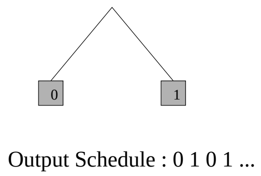

Figure 4.3: Scheduling tree (1 stream).

ity. Rate allocated by the scheduler r_i is hence represented by an array of g bits $b_{i1}, b_{i2}, b_{i3}, \ldots, b_{ig}$, and $r_i = \sum_{k=1}^{g} b_{ik} 2^{-k}$. Another assumption is that sum of rates from input streams is exactly equal to the output link rate which is 1. In order to achieve this, we add a null stream (or stream labelled 0), with an appropriate rate such that the sum of the rates of all the streams (including the null stream) becomes equal to 1. In the output schedule, whenever the null stream occurs, either the link is left idle or a cell from non real-time class (for example the best-effort traffic) is scheduled, or a cell from the "next stream" is scheduled.

4.3.1 Basic Scheduler

We now describe the scheduler, initially by examples and later with algorithmic description. The scheduler is constructed in the form of a binary tree (also referred as scheduling tree). We defer the method of constructing the scheduling tree to a later stage. Let us begin with some simple examples to illustrate the basic idea of a RRR scheduler. Upon startup, the scheduler has no incoming stream. The scheduling tree is initialized to a single node as shown in Figure 4.2. The output schedule is simply $0, 0, 0, \ldots$.

On the addition of a new constant stream of rate 0.5 the root node of the tree *splits* into two nodes as shown in Figure 4.3. The output schedule in this case is $0, 1, 0, 1, \ldots$.

As a new stream (stream 2) of rate 0.25 is added, a node at level 1 is *split* into

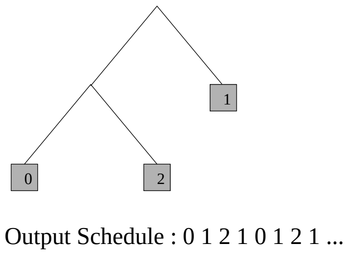

Figure 4.4: Scheduling tree (2 streams).

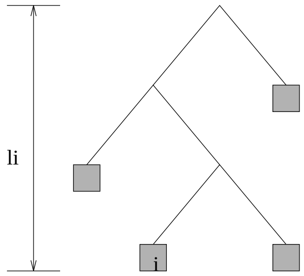

Figure 4.5: Scheduling tree (general case 1).

two nodes at level 2 and the tree takes the shape as shown in Figure 4.4. In this case, the output schedule of the RRR would be $0, 1, 2, 1, 0, 1, 2, 1 \ldots$.

A more general case is depicted in Figure 4.5, where stream i has a rate of 2^{-l_i}. In this case, a leaf at level l_i of the scheduling tree is allocated to stream i. If no such leaf exists, then a node labelled 0 at largest level $l < l_i$ is split into two leaves labelled 0 at level $l+1$. If $l+1 < l_i$ then one of the newly formed leaf is further split. Such split is carried out until two leaves at level l_i are created. One of these leaves is allocated to stream i. The algorithm for constructing the tree (adding stream) is isomorphic to the binary subtraction algorithm. Similarly, the procedure of deleting stream is isomorphic to the binary addition algorithm.

In the most general case, as shown in Figure 4.6, a node of the scheduling tree at level l_i is allocated to a stream if and only if l_ith bit in the stream's rate representation is 1. In other words, if the rate of a stream is equal to $\sum_{k=1}^{g} b_k 2^{-k}$, then a leaf of level k will be allocated to the stream if and only if b_k is equal to 1.

Given a set of rates, it is possible to construct a tree which satisfies the above described property. The tree has two kind of nodes, internal nodes and leaf nodes. A

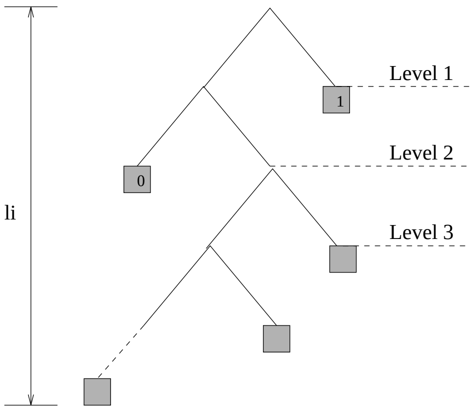

Figure 4.6: Scheduling tree (general case 2).

stream number is associated with each leaf node (in other words, the leaf is allocated to the stream). There is a bit associated with each internal node which keeps track of its subtree visited last. This bit is initialized to 0. In order to schedule a cell, the scheduler starts from the root of the tree and descends down to a leaf, scheduling cells from the stream number associated with the leaf. If the bit associated with an internal node is zero, the scheduler traverses its left subtree, otherwise it traverses the right subtree. In any case, the scheduler flips the bit associated with each internal node it visits. Thus the first cell would be scheduled from stream associated with the left most leaf and the second cell will be from the stream associated with the left most leaf of the right subtree of the root. Figure 4.7 gives the algorithmic description of the scheduler.

It may be noted that the service rate of a node m at level l_m is equal to 2^{-l_m}. A node m is visited every 2^{l_m} time units. A cell from the stream associated with leaf m is periodically scheduled in the interval of 2^{l_m} time units.

We now define two terms, level and phase, associated with each node of the tree.

Definition 4.3.1 (Level) *Level $l(m)$ of node m is defined as the distance of the node from the root of the tree.*

The level of the root is zero. The level of a child is one more than that of its parent.

```
Scheduler () {
        for_each(node) /* Initialization */
                node.bit = 0;
        for_ever
                schedule_one_cell (root);
}

schedule_one_cell (node) {
        if (node == leaf)
                output (node.stream_number);
        else_if (node.bit == 0) {
                node.bit = 1;
                schedule_one_cell (node.left_child);
        }
        else_if (node.bit == 1) {
                node.bit = 0;
                schedule_one_cell (node.right_child);
        }
}
```

Figure 4.7: The RRR scheduling algorithm.

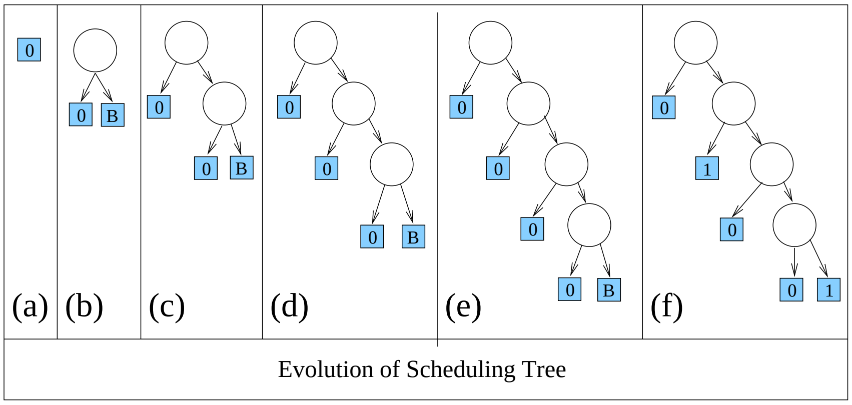

Figure 4.8: Adding stream 1 of rate 0.0101.

Definition 4.3.2 (Phase) *Phase $\phi(m)$ of a node m is the first time instant at which the node is visited.*

The phase of the root node is 0. The phase of the left child (m) of a node (n) is equal to that of the parent node. The phase of the right child (m) of node (n) is $2^{l(n)}$ more than that of the node. Formally

$$\phi(m) = \begin{cases} \phi(n) & \text{if } m \text{ is left child of } n \\ \phi(n) + 2^{l(n)} & \text{if } m \text{ is right child of } n \end{cases}$$

A cell from the stream associated with a leaf m is scheduled at times $\phi(m), \phi(m) + 2^{l(m)}, \phi(m) + 2 * 2^{l(m)}, \phi(m) + 3 * 2^{l(m)}, \ldots$. Note that if more than one leaf of scheduling tree are allocated to a stream, there is one such sequence corresponding to each allocated leaf. The time instants at which cells of a stream are scheduled is given by the union of these sequences.

4.3.2 Adding and Deleting Streams

Adding a stream with a given allocated rate is isomorphic to the deletion algorithm for binary numbers. One can view the addition of a stream with rate r as taking away r units of rate from the null stream. Similarly deleting a stream of rate r is isomorphic to binary addition, as r units of rate become free and get added to the rate of the null stream.

The scheduling tree is modified dynamically as streams are added or removed. However any modifications to the scheduling tree must be made carefully so that the delay bounds of the existing streams are not violated. We now define the property of unfragmented rate allocation which must be satisfied by every stream.

Definition 4.3.3 (Unfragmented rate allocation) *A stream i is said to have unfragmented rate allocation if and only if for any level l of the scheduling tree, there is at most one node allocated to the stream i.*

The basic idea is to ensure all the streams (including the null stream) have unfragmented rate allocation each time a stream is added or deleted from the scheduling tree. Unfragmented rate allocation bounds the number of leaves allocated to a particular stream. If the height of the scheduling tree is k and a stream has unfragmented rate allocation, then at most k nodes in the scheduling tree can be allocated to the stream. We will show that the number of leaves is directly related to delay and fairness properties. It also limits the size of the scheduling tree, and hence the amount of state and complexity of implementation.

To add a stream i with the requested rate r_i, the first step is to check if the requested rate is available. If the link does not have adequate rate, the requested stream is denied to be added to the scheduling tree. If the link has adequate rate, i.e., $r_i \leq r_0$, then we proceed further.

Assume

$$r_i = \sum_{k=1}^{g} b_{ik} 2^{-k}$$

The procedure begins with the least significant bit ($k = g$ *downto* 1) of the requested rate, from a node at the largest level. If b_{ik} is zero, no actions need to taken and k is simply decremented. If both b_{ik} b_{0k} are equal to 1, then the node at level k previously allocated to the null stream is allocated to the current stream i. This requires simple relabelling in the scheduling tree.

However, if $b_{ik} = 1$ and $b_{0k} = 0$, a *borrow* operation is performed from level k of the scheduling tree. During this operation, we move up to level $k-1$ in the scheduling tree and find if there is a leaf allocated to the null stream at $k-1$ level. If the answer is no, we move up the scheduling tree again till such a leaf is found. It is always possible to find such a leaf if $r_i \leq r_0$ and the null stream has unfragmented rate allocation. After a leaf allocated to the null stream at level k' is found, it is *split* into two children at level $k'+1$. One of the children is allocated to the null stream while the other child is used for performing the *split* operation again. This process is continued till level k is reached. At level k, one of the two new children is allocated to stream i and the other to the null stream.

The above procedure is continued till the root of the scheduling tree is reached. r_0 is decremented by r_i to update the residual bandwidth ($r_0 = r_0 - r_i$). It may be noted that at this stage, the scheduling tree has unfragmented rate allocation for all the streams.

The above procedure of adding streams is illustrated by examples in Figure 4.8, Figure 4.9 and Figure 4.10. In these examples, it is assumed that 4 bits are used to represent a rate.

Evolution of the scheduling tree when stream 1 of rate 0.0101 is added for scheduling on an idle link is shown in Figure 4.8(a) to (f). Initially there is no stream, so all the bandwidth is allocated to the null stream. The scheduling tree is single leaf as shown in (a). Since 4th bit of rate of stream 1 is 1, a node at level 4 must be allocated. Since there is no leaf allocated to the null stream at level 4, the borrow operation is performed. The first leaf allocated to the null stream is the root itself. Thus the root is split into two children at level 1. One of the children is allocated to the null stream and the other is marked with B representing the borrow operation, as shown in (b). The node marked B is split again as shown in (c). This borrow operation is carried as shown in (d) and (e). As the leaf node in (d) is split again, two leaves at level 4 associated with the null stream are created. One of the leaf is allocated to stream 1. Now there is a leaf at level 2 allocated to the null stream. This leaf is allocated to stream 1 resulting in the scheduling tree as shown in (f).

Figure 4.9 and Figure 4.10 are self explanatory as the borrow operation is not needed in allocating the nodes, while adding stream 2 of rate 0.0100 and stream 3 of rate 0.0011.

To delete stream i, all its allocated nodes are re-allocated to the null stream.

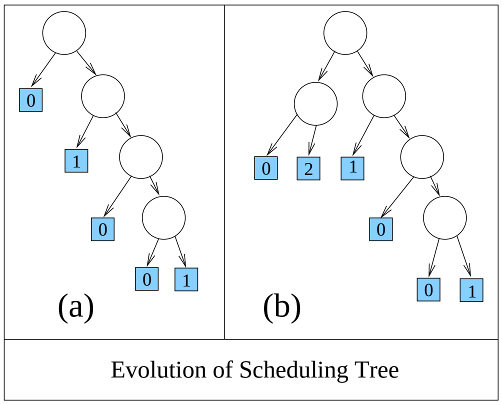

Figure 4.9: Adding stream of 2 rate 0.0100.

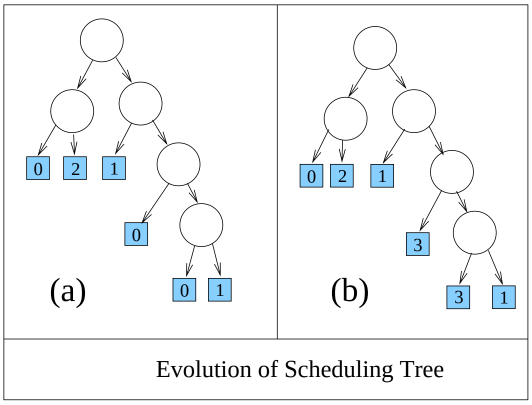

Figure 4.10: Adding stream 3 of rate 0.0011.

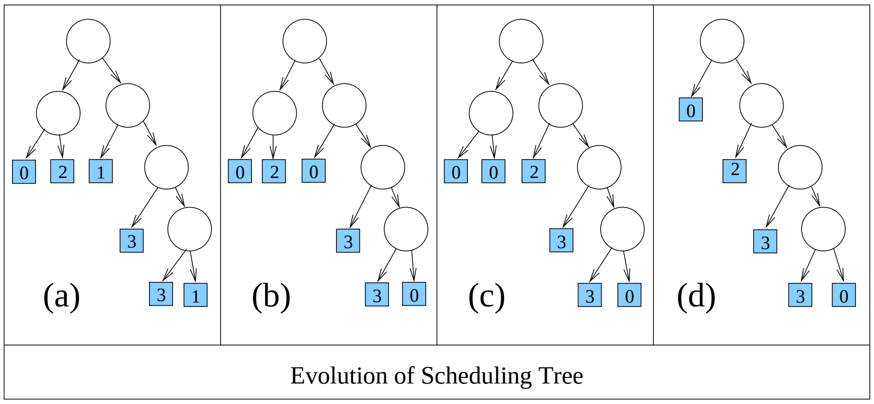

Figure 4.11: Deleting stream 1.

At this stage, the null stream may have fragmented rate allocation. In order to unfragment the rate allocation of the null stream, the inverse of the split (*join*) and borrow operation (*carry*) is performed.

If there are two nodes at the same level assigned to the null stream, they should be joined together to form a parent at a higher level. The join operation can be applied only on siblings (two nodes with a common parent). Assume that the n_1 and n_2 are the nodes allocated to the null stream at the same level say l. If n_1 and n_2 are siblings then the join operation is directly applied and the two nodes are replaced by a leaf node at level $l-1$. In case n_1 and n_2 are not siblings. Let n_3 be the sibling of n_1. Now the nodes n_2 and n_3 are swapped. This operation is called *phase exchange*[1]. After the phase exchange, n_1 and n_2 become siblings and the join and carry operation is performed to unfragment rate allocation of the null stream.

The above procedure is carried out from the largest level to the root of the scheduling tree, until the rate allocation of the null stream becomes unfragmented. Finally r_0 is updated to the value of the currently available bandwidth ($r_0 = r_0 + r_i$).

In continuation with the examples described in Figure 4.8 to Figure 4.10, we illustrate deleting stream 1 in Figure 4.11. The initial scheduling tree is shown in (a). In order to delete stream 1 all the leaves allocated to it are allocated to the null stream as shown in (b). Now there are two leaves at level 2 allocated to the null stream. Thus the null stream is now fragmented. In order to unfragment the null stream, the phase exchange between two leaves (one allocated to stream 2 and the other to stream 0) at level 2 is carried out, resulting in the scheduling tree as shown in (c). Finally the two siblings allocated to the null stream are merged together to move up, resulting in a leaf allocated to the null stream at level 1 as shown in (d). Now all the streams have unfragmented rate allocation.

Note that the procedure for construction of scheduling tree when rates are repre-

[1]The phase exchange has to be carried out carefully such that it doesn't affect the delay bound and other properties of the stream associated with n_2. The phase exchange is carried out in two steps. Firstly node n_3 allocated to null stream is reallocated to the stream associated with n_2. At this stage the stream gets more share of link rate. If current time is t then node n_2 is reallocated to null stream at time $t + 2^l$. This deallocation can be done asynchronously as the second step.

sented in floating point, is an obvious extension of the method described above.

4.3.3 Allocation for Floating Point Rates

In this section we will discuss the rate allocation for the scheduling tree when the rate is represented in the form of floating point. Let the rate $r \leq 1$ be represented in simple binary floating point with g bits of mantissa and e bits of unsigned exponent. Let E be the value of exponent. Now $r = 2^{-E} \sum_{k=1}^{g} \frac{b_k}{2^k}$. The rate allocation for this rate is possible if the height of the scheduling tree is $E + c$. If $E_{\max}$ is the maximum value which the exponent can take, then the rate allocation as discussed earlier is possible provided the height of the scheduling tree is $E_{\max} + c$.

It is important to note that even if the height of the scheduling tree has been increased to a larger value, the number of nodes allocated for a stream is still bounded by g, the mantissa (granularity) of rate representation. Keeping this fact in mind is possible to implement the scheduler in hardware as well as software with reasonable complexity. The implementation of the floating point rate representation will be very similar to that of the basic scheduler. The bounds and other properties of the scheduler change marginally if the floating point rate allocation is carried out. In the rest of the chapter, the fixed point rate representation is used to simplify descriptions and proofs.

4.3.4 Over allocation and Bit Rate Granularity Tradeoff

One can reduce the granularity of rate allocation by reducing the number of bits to represent the mantissa of the normalized rate. The delay and fairness bounds are directly proportional to the rate granularity. The size of the scheduling tree, and the time needed to make a scheduling decision is also proportional the rate granularity. Reducing the rate granularity results in better delay and fairness properties and faster and more efficient implementation. The cost incurred in reducing the rate granularity is rate over allocation. Consider g bits of rate granularity. In order to satisfy the rate constraint, the rate allocated to a stream must be greater than or equal to the rate of the stream. However, since only finite precision floating point rate representation is used, not all possible rates can be represented in the format. Therefore, the allocated rate should be the smallest floating point number (using g bits of mantissa) greater than or equal to the rate of the stream. This results in an over allocation in rate. In the worst case, the fraction of rate over allocated is bounded by 2^{1-g}. Thus using 1 bits of mantissa to represent rate results in at most 100 percent over allocation in rate. This is permissible if the best-effort traffic is expected to take more than 50 percent of the link rate. However, for a rate granularity of 8 bits, this over allocation is less than one percent.

4.3.5 Work-Conserving RRR

We refer to the scheduler discussed before as *basic scheduler* or *basic RRR*. The obvious extension of the basic scheduler is to a work-conserving scheduler.

In the basic RRR scheduling, the time slots which a stream gets is fixed once the allocation of leaves for that stream has been completed. The slots are independent of traffic on other streams. It might happen that a scheduling slot of a stream arrived but there is no pending cell to be scheduled. In such a case no cell will be scheduled if the RRR is used. The work-conserving RRR will however schedule a cell from the next non back-logged stream. Figure 4.12 describes the work-conserving variant of RRR.

4.3.6 Scheduling Variable Size Packets

In order to schedule variable sized packets, each internal node of the scheduling tree stores the amount of service given to the node. The service given to a subtree is defined to be the sum of service given to its leaves. Thus, if the link rate is l and the scheduler is always busy, then the service given to the root node in time t is lt. While descending down the tree, child with the smallest service given is selected. After reaching the leaf, a packet from the stream corresponding to the leaf is scheduled and the corresponding service variable is incremented by the packet size, at each node in the path from root to the leaf. In case of the non work-conserving version of RRR, it there is no packet of the stream corresponding to the leaf, a packet from the best-effort

```
Scheduler () {
        for_each(node) /* Initialization */
                node.bit = 0;
        for_ever
                schedule_one_cell (root);
}

schedule_one_cell (node) {
        if (node == leaf) {
                if (there are no pending cells)
                  return; /* Scheduler is idle */
                if there is a pending cell on stream (node.stream_number)
                  output (node.stream_number);
                else
                  schedule_one_cell (root)
        else_if (node.bit == 0) {
                node.bit = 1;
                schedule_one_cell (node.left_child);
        }
        else_if (node.bit == 1) {
                node.bit = 0;
                schedule_one_cell (node.right_child);
        }
}
```

Figure 4.12: Work-Conserving RRR.

```
Scheduler () {
        for_each(node) /* Initialization */
                node.service = 0;
        for_ever
                schedule_one_packet (root);
}

schedule_one_packet (node) {
        if (node == leaf) {
                if there are no pending cells on (node.stream_number)
                        service = S_max; /* To indicate virtual service */
                else {
                        output (node.stream_number);
                        service = node.stream_number.packet_size;
                }
        else_if (node.left_child.service ≤ node.right_child.service)
                service = schedule_one_packet (node.left_child);
        else
                service = schedule_one_packet (node.right_child);
        node.service += service;
        return service;
}
```

Figure 4.13: Variable packet size scheduler (work-conserving).

queue is scheduled and the corresponding service variables are incremented by the size of the best-effort packet. In case of work-conserving variant, the service variables in the path are incremented by the maximum packet size (to indicate virtual service) and the tree is traversed again from root to get the next packet to be scheduled. Figure 4.13 gives an algorithmic description of the work-conserving version of the variable size packet scheduler.

It is not necessary to maintain the absolute value of the service given to a node from the beginning. Only, the difference between the services given to left and right subtree of a node need to be maintained. We will see in Section 4.4.5 that the difference between the service given to two children of a node is bounded by S_{max}. Therefore, an integer in the range $[0, 2S_{max} + 1]$ is sufficient at each node.

4.3.7 Discussion

In the last section we presented the main idea behind the scheduler. The link bandwidth is partitioned into two equal halves by splitting the node at root of the scheduling tree into two nodes at level 1. While scheduling, these two nodes are visited in round robin order. Both the nodes at level 1 can be further split into two nodes each at next level, further partitioning the link bandwidth. Continuing in this manner, the link bandwidth can thus be partitioned hierarchically giving rise to a scheduling tree. Traversal of nodes of the tree becomes *recursive round robin*.

The concept of scheduling tree is abstract. It is not always necessary to construct the scheduling tree for generating the schedule. Hardware implementation sketched in section 4.7 does not explicitly store the scheduling tree data structure. Therefore, the time complexity of the scheduler should not be judged by the time complexity of algorithm previously described in Figure 4.7. Several implementations of the algorithm are possible and the tradeoff is discussed in section 4.7. We believe that efficient hardware implementation of the scheduler can schedule one cell per clock cycle. The amount of hardware needed scales linearly with the rate granularity g.

Symbol	Meaning
$A_i(t)$	Number of cells of stream i arriving in interval $[0,t]$
$S_i^j(t)$	Number of cells of stream i scheduled at node j in time $[0,t]$
$B_j(t)$	Backlog of stream i at time instant t
$b_j(t)$	Burstiness function of stream i
r_i^j	Normalized rate allocated to stream i at link j
c_i^j	Count of number of 1s in r_i^j.

Table 4.1: Notations.

4.4 Main Properties of RRR Scheduler

We will show that all the properties of RRR scheduler scale linearly with granularity of rate allocation, g. Specifically, for a stream i all the properties depend on c_i, which is count of number of ones in binary representation of r_i. c_i is bounded by granularity of rate allocation, g.

The notations used are summarized in Table 4.1. Let $S(t)$ be equal to the number of cells of the stream scheduled by the basic RRR scheduler in the interval $[0,t]$. Then

$$S_i(t) \leq \sum_{k=1}^{g}[\lfloor\frac{t-\phi(b_{ik})}{2^k}\rfloor+1]b_{ik} \tag{4.1}$$

where b_{ik} is k^{th} bit in the normalized rate (r_i) allocated to stream i, and $\phi(b_{ik})$ is phase of corresponding the k^{th} bit. The right hand side of equation 4.1 is involved in all the proofs. Intuitively, the contribution to $S_i(t)$ is only from terms in the summation where b_{ik} is non zero. For each k such that b_{ik} is equal to 1, cells are scheduled at time instants $\phi(b_{ik}), \phi(b_{ik})+2^k, \phi(b_{ik})+2*2^k, \ldots$. Summing this over all values of k gives the equation 4.1.

4.4.1 Delay Bound for Single Node

For a stream, say i which is continuously backlogged in the interval $[0,t]$ we have

$$S_i(t) \geq \sum_{k=1}^{g}[\lfloor\frac{t-\phi(b_{ik})}{2^k}\rfloor+1]b_{ik} \tag{4.2}$$

Equation 4.2 is true for both basic RRR scheduler and work-conserving RRR. The equation is true even in the presence of phase exchanges described in section 4.3.2. This gives the following upper bound (d_i) for delay.

Theorem 4.4.1 (Single node delay bound) *The delay bound d_i of a stream i, which is compliant to the leaky bucket parameters (σ_{imax}, r) is given by:*

$$d_i \leq \frac{1}{r_i}(\sigma_{imax} + c_i) \tag{4.3}$$

where r_i is the rate allocated to the stream sich that $r_i \geq r$ and c_i is the count of number of ones in the normalized binary representation of rate r_i.

We use the notation shown in Table 4.1 in our proof. Let $b(t)$ be a bounding burstiness function of the stream under consideration (see Chapter 2). This function was defined in [24] and in Chapter 2. Let r be the allocated rate of RRR scheduler for scheduling the compliant stream. The rate r should be chosen carefully while performing admission control as it might affect the network utilization. Some guidelines to choose r are described in Chapter 2. For constant rate streams r could be chosen which is close to the bit rate of the stream. The rate r of the compliant stream can take arbitrary value in $[0, 1)$. Given rate r, it is desirable to pick $r_a > r$ such that the number of times 1 occurs in the g bit binary representation of r is minimized without adding significantly to the overhead. Such a selection can further minimize the jitter introduced in the stream by the scheduler.

Without loss of generality, we assume that the busy period, in which a cell incurs maximum delay, starts at time $t = 0$. Also assume that the value of time t is limited to this busy period. Let $A(t)$ be the number of cells of a compliant stream arrived in the interval $[0, t]$. Assume that the cell arrived at time t' suffers the maximum scheduling delay.

Let d be the delay of cell arriving at time t'. This implies that all the traffic arriving in the interval $[0, t']$ is serviced exactly till time $t + d$. Therefore,

$$A(t') = S(t' + d) \tag{4.4}$$

Substitution of $S(t)$ of Eq.(4.2) in Eq.(4.4), after some manipulation, yields

$$\begin{aligned}
& \textstyle\sum_{k=1}^{g}[\lfloor \frac{t'+d-\phi(b_k)}{2^k} \rfloor + 1]b_k && \leq A(t') && (4.5)\\
\Rightarrow \quad & \textstyle\sum_{k=1}^{g}\lfloor \frac{t'+d}{2^k} \rfloor b_k - c && \leq A(t') && (4.6)\\
\Rightarrow \quad & rd + rt' - c && \leq A(t') && (4.7)
\end{aligned}$$

Since $A(t) \leq b(t)$ we get

$$rd \quad \leq \quad b(t') - rt' + c \tag{4.8}$$

The burst size is defined as $\sigma_{\text{max}} = \max_t(b(t) - rt)$. Substituting σ_{max} in above equation we get

$$\begin{aligned}
rd & \leq \sigma_{\text{max}} + c && (4.9)\\
\Rightarrow \quad d & \leq \frac{1}{r}[\sigma_{\text{max}} + c] && (4.10)
\end{aligned}$$

After de-normalization, when the unit of rate is cells per second and the unit of σ_{max} is number of cells, the above equation gives the delay bound in seconds.

The worst case delay (for compliant streams) is given by Eq.(4.10). It may be noted that Eq.(4.2) gives a lower bound on $S(t)$. Therefore, the same proof applies to the work-conserving RRR and the RRR with dynamic addition and removal of streams (This inequality would hold true in *phase shift* operation needed to unfragment the rate of the null stream).

The term $\frac{\sigma_i \text{max}}{r_i}$ is because of inherent burstiness of the traffic being scheduled, and the term $\frac{c_i}{r_i}$ is because of the non uniformity in the output schedule.

Compare this bound with that of "optimal fair scheduler" WFQ which is $\frac{1}{r_i}(\sigma_i\text{max}+1)$. Typically the burst size is large as compared to the parameter c_i. For instance assume a stream of MPEG video of rate 5 Mbps ($r = 11.8K$ ATM cells per second), and burst size of $10KB$ (189 cells) is scheduled by RRR scheduler with 16 bits of rate allocation. The total delay bound would be $17.2ms$. The corresponding bound for WFQ scheduler is $16.1ms$.

4.4.2 Delay Bound for Multiple Nodes

Theorem 4.4.2 (Network delay bound) *If a stream compliant with leaky bucket parameters* (σ_{max}, r) *passes through* N *nodes with rate allocation of* $r^j \geq r$ *at node* j*, then its end-to-end delay bound* d *is given by:*

$$d \leq \frac{1}{r}(\sigma_{max} + 2\sum_{k=1}^{N} c^k) \tag{4.11}$$

where c^j *is the count of number of ones in the normalized binary representation of* r^j*.*

Consider a stream across N nodes in the network. Assume basic RRR scheduler is being used at each node. Let the rate allocated to the stream at various nodes be $(r^1, r^2, \ldots, r^N)$. For technical reason we assume that $r^1 \leq r^2 \leq r^3 \ldots \leq r^N$[2]. We prove delay bound by first showing that output stream is compliant to a leaky bucket parameter. The delay bound for single node case is then used to bound the delay at each node.

A stream scheduled by basic RRR scheduler complies to leaky bucket parameter (c, r). The number of cells arriving in time interval $(0, t]$ is equal to $S(t)$. For basic RRR,

$$S(t) \leq \sum_{k=1}^{g}[\lfloor\frac{t - \phi(b_k)}{2^k}\rfloor + 1]b_k$$

Simplification of above yields:

$$S(t) \leq rt + c$$

Therefore, the output stream complies to leaky bucket with parameter (c, r). The output stream of first node complies to leaky bucket with parameters (c^1, r^1). So the delay at second node is bounded by $\frac{1}{r^2}(c^1 + c^2)$. Summing this over all k, after some simplifications yields an end to end delay bound of

$$\frac{1}{r^1}(\sigma\text{max} + 2\sum_{k=1}^{N} c^k)$$

In many situations, this bound is close to that of network of GPS schedulers as the term $\frac{\sigma\text{max}}{r}$ dominates the expression.

[2]We show in next chapter that tighter network delay bounds $(\frac{1}{r}(\sigma\text{max}+\sum_{k=1}^{N} c^k))$ can be obtained using the theory of latency-rate servers [117]. The assumption of monotonically increasing rate along the path is also not required.

4.4.3 Buffers Needed at Intermediate Nodes

The output stream of basic RRR scheduler is smooth. As a result a bound on number of buffers needed at an intermediate node to guarantee zero cell loss is obtained.

At node k, maximum backlog $B^k(t)$ is bounded by $S^k(t) - A^k(t)$. The arrival $A^k(t)$ satisfies

$$A^k(t) \leq r^{k-1}t + c^{k-1}$$

For a busy period beginning at $t \leq 0$, the output $S^k(t)$ satisfies

$$S^k(t) \geq r^k t - c^k$$

If $r^k = r^{k-1}$ then

$$B^k(t) \leq c^{k-1} + c^k$$

The maximum backlog is also a bound on the buffer required.

At link j, $c^{j-1} + c^j$ cells of buffer is needed to guarantee zero cell loss. Thus, if 20 bits of rate representation is used, then an average of 20 cells of buffer per stream would be needed at intermediate nodes.

4.4.4 Fairness Properties

In case a link is under-utilized by streams scheduled on it, the remaining link capacity should be distributed to active (or backlogged) streams fairly, i.e. in proportion of their allocated rate [100]. Ideally, if stream i and stream j are continuously backlogged in the interval $[t_1, t_2]$ then amount of service in the interval should be related by:

$$\frac{S_i(t_1, t_2)}{r_i} = \frac{S_j(t_1, t_2)}{r_j} \tag{4.12}$$

In practice it is impossible to achieve ideal fairness because of cell boundaries. So a measure of fairness, called "service fairness index" (SFI) is defined to quantify the fairness.

$$SFI = \max_{t_1, t_2} \mid \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \mid \tag{4.13}$$

Small values of SFI correspond to more fairness. $SFI = 0$ corresponds to ideal fairness.

Theorem 4.4.3 (Service fairness index) *The service fairness index (SFI) for work-conserving RRR is given by:*

$$\frac{c_i}{r_i} + \frac{c_j}{r_j} \tag{4.14}$$

If two streams i and j are continuously backlogged in the interval $(t_1, t_2]$ and the rate allocated to them is r_i and r_j and $S_i(t_1, t_2)$ and $S_j(t_1, t_2)$ is the number of cells of stream i and stream j scheduled by the RRR scheduler in the interval $(t_1, t_2]$ then SFI is defined as:

$$\mathrm{SFI} = \max_{t_1, t_2} \mid \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \mid$$

Without loss of generality, assume that SFI is maximized when $t_1 = 0$. Let $S_i(t)$ denote $S_i(0, t)$. Let $v(t)$ be the number of leaves of the scheduling tree visited in time $(0, t]$. $v(t)$ is similar to the notion of virtual time used in the context of fair schedulers (like weighted fair queueing). Note that $v(t) = t$ is all the streams are continuously backlogged in the interval $[0, t]$. We first lower bound $\frac{S_i(t)}{r_i}$ as follows:

$$\begin{aligned}
S_i(t) &= \sum_{k=1}^{g} [\lfloor \frac{v(t) - \phi(b_{ik})}{2^k} \rfloor + 1] b_{ik} && (4.15)\\
&\geq \sum_{k=1}^{g} [\frac{v(t) - \phi(b_{ik})}{2^k}] b_{ik} && (4.16)\\
&\geq v(t)[\sum_{k=1}^{g} \frac{b_{ik}}{2^k}] - c_i && (4.17)\\
&\geq v(t) r_i - c_i && (4.18)\\
\Rightarrow \frac{S_i(t)}{r_i} &\geq v(t) - \frac{c_i}{r_i} && (4.19)
\end{aligned}$$

Similarly, we upper bound $\frac{S_i(t)}{r_i}$ as

$$\frac{S_i(t)}{r_i} \leq v(t) + \frac{c_i}{r_i} \tag{4.20}$$

Equations 4.19 and 4.20 give

$$\begin{aligned}
\Rightarrow \mid \frac{S_i(t)}{r_i} - \frac{S_j(t)}{r_j} \mid &\leq (\frac{c_i}{r_i} + \frac{c_j}{r_j}) && (4.21)\\
\Rightarrow \mathrm{SFI} &= (\frac{c_i}{r_i} + \frac{c_j}{r_j}) && (4.22)
\end{aligned}$$

Equation 4.22 gives a bound on service fairness index of work-conserving RRR. The bound on SFI increases linearly with increasing granularity, as c_i and c_j are bounded by g.

Another measure of fairness is based on the worst case delay for clearing the backlog of a stream's queue. A scheduler has worst case fair index (WFI) of $\frac{C_i}{r_i}$ for stream i if C_i is the smallest number satisfying the equation:

$$S_i(t_1, t_2) \geq (t_2 - t_1)r_i - C_i \tag{4.23}$$

The normalized worst case fair index would be C_i. This measure is important in hierarchical scheduling [129]. The delay bound of hierarchical scheduler increases with increasing WFI.

Theorem 4.4.4 (Worst case fairness index) *The worst case fariness index (WFI) of work-conserving RRR scheduler is bounded by $\frac{c_i}{r_i}$.*

If C_i is the smallest number satisfying the equation:

$$S_i(t_1, t_2) \geq (t_2 - t_1)r_i - C_i \tag{4.24}$$

then WFI $= \frac{C_i}{r_i}$.

Since $v(t) \geq t$, Eq.(4.18) yields

$$S_i(t) \;\; \geq \;\; tr_i - c_i \tag{4.25}$$

So WFI for stream i is bounded by $\frac{c_i}{r_i}$. Normalized WFI is bounded by c_i, which in turn is bounded by g, the granularity of rate allocation.

4.4.5 Bounds for Variable Packet Size Scheduler

The most important property for proving bounds for the variable packet size scheduler is that the service given to a node is divided almost equally among its children. The following theorem describes this property for the binary RRR scheduler.

Theorem 4.4.5 *If $S(t_1, t_2)$ is the amount of service given to a node in interval $[t_1, t_2]$ and $S_l(t_1, t_2)$ and $S_r(t_1, t_2)$ are the portion of this service given to its left and right child respectively, then:*

$$\frac{1}{2}S(t_1, t_2) - S_{max} \leq S_l(t_1, t_2) \leq \frac{1}{2}S(t_1, t_2) + S_{max} \tag{4.26}$$

$$\frac{1}{2}S(t_1, t_2) - S_{max} \leq S_r(t_1, t_2) \leq \frac{1}{2}S(t_1, t_2) + S_{max} \tag{4.27}$$

where S_{max} is the maximum size of packets in the network.

Let $S(t), S_l(t)$ and $S_r(t)$ represent the amount of service given to a node and its left and right child respectively in the interval $[0, t]$. Note that if there is no packet waiting to be served at a leaf, then we give some virtual service to the leaf (and its parent nodes) and either serve a packet from the best-effort queue, or move to the next schedule. The service $S(.)$ represents the sum of real and virtual service given to the nodes. The amount of service given to a node is divided among its left child and right child. Therefore,

$$S(t) = S_l(t) + S_r(t) \tag{4.28}$$

If the node is visited during tree traversal, its child with less service received will also be visited. This implies, the service given to a node will always be passed to the child which has received less service. The maximum non preemptable unit of this service is S_{max} and in the beginning (at $t = 0$) every node is initialized to indicate equal (zero) service received. Thus we have the following lemma.

Lemma 4.4.1 *The absolute difference between service given to left and right child of a node is bounded by the maximum packet size. Mathematically:*

$$\mid S_l(t) - S_r(t) \mid \leq S_{max} \tag{4.29}$$

We prove this lemma by induction. Let τ_k be the time instant just after the node is visited for the kth time ($\tau_0 = 0$). We apply induction on k. The base case when $\tau = 0$ is trivially satisfied since $s_l(0) = s_r(0) = 0$. Assume that the induction hypothesis is true for a given k. Thus:

$$\mid S_l(\tau_k) - S_r(\tau_k) \mid \leq S_{max} \tag{4.30}$$

Assume that

$$S_l(\tau_k) > S_r(\tau_k) \tag{4.31}$$

Now, when the parent node is visited for the $k+1$th time, the service is given to its right child because $S_r(\tau_k)$ is smaller. The maximum quantum of service is bounded by S_{max}. Therefore we have:

$$\begin{aligned}
& S_r(\tau_{k+1}) && \leq S_r(\tau_k) + S_{max} && (4.32)\\
& S_l(\tau_{k+1}) && = S_l(\tau_k) && (4.33)\\
\Rightarrow \quad & S_r(\tau_{k+1}) && \leq S_l(\tau_k) + S_{max} && (4.34)\\
& && = S_l(\tau_{k+1}) + S_{max} && (4.35)\\
\Rightarrow \quad & S_r(\tau_{k+1}) - S_l(\tau_{k+1}) && \leq S_{max} && (4.36)
\end{aligned}$$

From Eq. 4.30 and 4.31 we also have

$$S_l(\tau_k) - S_r(\tau_k) \leq S_{max} \tag{4.37}$$

Since $S_l(\tau_{k+1}) = S_l(\tau_k)$ and $S_r(\tau_{k+1}) > S_r(\tau_k)$ we have

$$S_l(\tau_{k+1}) - S_r(\tau_{k+1}) \leq S_{max} \tag{4.38}$$

Combining Eq. 4.36 and 4.38 we get

$$\mid S_l(\tau_{k+1}) - S_r(\tau_{k+1}) \mid \leq S_{max} \tag{4.39}$$

The case when $S_l(\tau_k) < S_r(\tau_k)$ is symmetric and can be proven is exactly the same way. When $S_l(\tau_k) = S_r(\tau_k)$ then the tie is broken arbitrarily and service is given to one of the children. The maximum amount of service is bounded by S_{max}, therefore we can again claim that $\mid S_l(\tau_{k+1}) - S_r(\tau_{k+1}) \mid \leq S_{max}$. This shows that if induction hypothesis is true for any k, it is also true for $k+1$. Therefore it is true for all values of k. This completes proof of the lemma.

The following lemma will establish Theorem 4.4.5.

Lemma 4.4.2 *The amount of service given to a node is divided among its left child and right child according to the following equation:*

$$\frac{1}{2}(S(t) - S_{max}) \leq S_l(t) \leq \frac{1}{2}(S(t) + S_{max}) \tag{4.40}$$
$$\frac{1}{2}(S(t) - S_{max}) \leq S_r(t) \leq \frac{1}{2}(S(t) + S_{max}) \tag{4.41}$$

There are two cases. In case $S_l(t) \geq S_r(r)$ then Eq. 4.29 may be written as:

$$S_l(t) - S_r(t) \leq S_{max} \tag{4.42}$$
$$\Rightarrow \quad S_r(t) \geq S_l(t) - S_{max} \tag{4.43}$$

Substituting this value of $S_r(t)$ in Eq. 4.28 we get:

$$S_l(t) + S_l(t) - S_{max} \leq S(t) \tag{4.44}$$
$$\Rightarrow \quad S_l(t) \leq \frac{1}{2}(S(t) + S_{max}) \tag{4.45}$$

Also since $S_l(t) \geq S_r(t)$ Eq. 4.28 gives:

$$2S_l(t) \geq S(t) \tag{4.46}$$
$$\Rightarrow \quad S_l(t) \geq \frac{1}{2}S(t) \tag{4.47}$$

Therefore

$$\frac{1}{2}S(t) \leq S_l(t) \leq \frac{1}{2}(S(t) + S_{max}) \tag{4.48}$$

Substituting $S_l(t)$ from Eq. 4.28 we get

$$\frac{1}{2}(S(t) - S_{max}) \leq S_l(t) \leq \frac{1}{2}S(t) \tag{4.49}$$

The above two equations may be rewritten in the form of Lemma 4.4.2. The other case when $S_l(t) \leq S_r(t)$ is symmetric and a similar proof applies. This completes of proof of Lemma 4.4.2.

The service given to any node in an interval $[t_1, t_2]$ may be written as:

$$S(t_1, t_2) = S(t_2) - S(t_1) \tag{4.50}$$
$$S_l(t_1, t_2) = S_l(t_2) - S_l(t_1) \tag{4.51}$$
$$S_r(t_1, t_2) = S_r(t_2) - S_r(t_1) \tag{4.52}$$

Using the above equations and Lemma 4.4.2 we get Theorem 4.4.5.

Using Theorem 4.4.5, successively on the descendents of a node, we can establish a relationship between the service given to a node and another node in its subtree.

Theorem 4.4.6 *The service $S(t_1, t_2)$ given to a node and the service given to another node $S_n(t_1, t_2)$ in its subtree are related by:*

$$S(t_1, t_2) - 2S_{max}(1 - \frac{1}{2^n}) \quad \leq S_n(t_1, t_2) \quad \leq S(t_1, t_2) + 2S_{max}(1 - \frac{1}{2^n}) \tag{4.53}$$

where n is the difference between the depths of the two nodes.

Note, if $n = 1$ then nodes have parent child relationship and Theorem 4.4.5 becomes a special case of the Theorem 4.4.6.

Therefore, the amount of service (real and virtual) given to stream i in interval $[t_1, t_2]$ is given by:

$$S_i(t_1, t_2) \geq \sum_{k=1}^{g} \frac{b_k}{2^k}(S(t_1, t_2) - 2S_{max}) \tag{4.54}$$

$$= (\sum_{k=1}^{g} \frac{b_k}{2^k})S(t_1, t_2) - 2c_i S_{max} \tag{4.55}$$

$$= \frac{r_i}{l} l(t_2 - t_1) - 2c_i S_{max} \tag{4.56}$$

$$= r_i(t_2 - t_1) - 2c_i S_{max} \tag{4.57}$$

If stream i is continuously backlogged in the interval $[t_1, t_2]$ then all its service received will be real service. Therefore Eq. 4.57 can be used to compute delay and fariness bounds for the variable packet size RRR scheduler. Note the term $2c_i S_{max}$ in the right hand side of Eq. 4.57. The properties like delay bound, latency, buffer bound and fairness properties all depend directly on this term. The latency of this scheduler is equal to $\frac{2c_i}{r_i} S_{max}$, the buffer bound at intermediate nodes is equal to $2c_i S_{max}$ and the SFI is $2(\frac{c_i}{r_i} + \frac{c_j}{r_j})S_{max}$ and WFI is $2\frac{c_i}{r_i} S_{max}$. Most of the properties of variable packet size scheduler are double of the corresponding properties of the fixed packet size scheduler.

Scheduler	Latency	SFI	WFI	Implementation
GPS [100]	0	0	0	Impossible
WFQ [26]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}$	$O(\max \frac{S_i}{r_i})$	$O(n)$	$O(n)$
WF2Q [8]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}$	$O(\max \frac{S_i}{r_i})$	$\frac{S_i}{r_i}$	$O(n)$
Virtual Clock [131]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}$	∞	∞	$O(\log n)$
SCFQ [52]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}(n-1)$	$O(\max \frac{S_i}{r_i})$		$O(\log n)$
SPFQ [118]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}$	$O(\max \frac{S_i}{r_i})$	$O(n)$	$O(\log n)$
WF2Q+ [129]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}$	$O(\max \frac{S_i}{r_i})$	$\frac{S_i}{r_i}$	$O(\log n)$
FFQ [116]	$\frac{S_i}{r_i} + \frac{S_{max}}{r}$	$O(F)$	$O(F)$	$O(\log n)$
HRR [68]	$O(F)$			O(1)
DRR [111]	$O(F)$	$O(F)$	$O(n)$	$O(1)$
RRR [44]	$g\frac{S_i}{r_i}$	$gO(\max \frac{S_i}{r_i})$	$g\frac{S_i}{r_i}$	$O(g)$
RRR-Variable	$2g\frac{S_i}{r_i}$	$2gO(\max \frac{S_i}{r_i})$	$2g\frac{S_i}{r_i}$	$O(g)$

Table 4.2: Comparison of properties of RRR.

4.4.6 Comparison with Other Scheduler

Table 4.2 lists the main properties of RRR along with that of other schedulers. The latency, SFI and WFI for GPS scheduler are all zero. The GPS is a hypothetical scheduler based on the fluid flow model which cannot be realized in practice. The WFQ is an approximation of GPS for scheduling discrete packets, but is not ideal for hierarchical scheduling, as it has a high WFI. The WF2Q has been shown to be the closest packet-by-packet approximation of GPS [8]. However, it is difficult to implement WF2Q in real practice as its implementation requires a simulation of the original GPS algorithm. An implementation of WF2Q may take $O(n)$ time to schedule a packet in the worst case. All these algorithms first stamp a packet with a tag (based on the simulation of the GPS) and then insert the packet in a sorted priority queue. The packets are scheduled in increasing order of their tags. The algorithm to calculate the timestamp takes $O(n)$ time in these algorithms. The Virtual Clock, self clocked fair queueing (SCFQ), start potential fair queueing (SPFQ) scheduler have a simpler algorithm for calculation of this tag, but these algorithms do not have good fairness properties. The WF2Q+ algorithm has a fast tag calculation algorithm and has good delay and fairness properties, but in order to select a packet, the scheduler needs to sort all the packets on the basis of their tags. This may take $O(\log n)$ time, in the worst case to schedule a packet which may be unacceptable. The frame based scheduling algorithms like frame based fair queueing (FFQ), hierarchical round robin (HRR), deficit round robin (DRR) are faster, but most of their delay and fairness properties are directly proportional to their frame size (F). This means the the frame size needs to be kept small for good delay and fairness properties. A small frame size limits the rate allocation granularity. In most of the cases the rate allocation granularity is reciprocal of the frame size. This may lead to under utilization of bandwidth due to rate over allocation. In case of basic RRR scheduler, the schedule is repeated after a period of 2^g (if fixed point rate representation is used). Thus it may be viewed as a frame based scheduler with a frame size of 2^g. All the properties of the RRR scheduler are proportional to the rate granularity (g), which may be viewed as the log of the frame size. RRR trades a little of speed for much improved delay and fairness properties as compared to the frame based schedulers. The basic operations in case of RRR are simple bit manipulations, as a result of which it is suitable for high speed implementation.

4.5 Variants of RRR

Now we are ready to introduce and discuss variants of RRR.

4.5.1 With k-way branching: k-ary RRR

In k-ary RRR scheduler, the scheduling tree is not binary. The degree of the scheduling tree is k. In order to schedule one cell, the tree is traversed recursively starting from the root node. Every non-leaf node contains an index which takes a value from 0 to $k-1$. The index is initialized to 0. For any internal node, the child pointed by the index of the node is recursively visited. The index of the node is incremented by 1 (mod k), each time the node is visited.

In case of the k-ary variant of the variable packet size RRR, while traversing the tree, the scheduler attempts to equalize the service given to all the children of a node. Thus, every time a node is visited, the child receiving the minimum service is selected for traversal. As a result the services given to any node in a given time interval, remain fairly close to its share.

The properties of the k-ary scheduler can be derived along the lines of binary scheduler. The normalized rate is now represented in base k. The delay bound, SFI and WFI are now proportional to the sum of digits in the normalized representation. In general, these bound are a factor $k - 1$ larger (in the worst case) than that of binary scheduler (assuming that scheduling trees of same depth are used). In certain special cases, for instance when the set of normalized rates are a power of $1/k$, the bounds for k-ary RRR may become better than that of binary RRR.

4.5.2 Generalized RRR

In generalized RRR, the degree of nodes at different levels of the tree can be different. For each level l there is a degree parameter d_l. Each internal node at level l has d_l children. The scheduling tree is traversed recursively as in the k-ary RRR scheduler, except that the parameter index of a node at level l can take a value from 0 to $d_l - 1$. The index at level l is incremented modulo d_l. The variable packet size scheduler always selects the child with minimum service received.

4.5.3 Recursive Tree Based Scheduling

This is the most general form of recursive scheduling. In this case the scheduling tree can be arbitrary. Each node in the tree contains the count of the number of its children. In order to schedule a cell the tree is traversed recursively. While visiting an internal node, its index is incremented modulo its number of children. The tree need not be static. New internal nodes of different degrees can be created when needed. The nodes at various parts in the tree can be brought together and combined into a single node. In order to schedule variable packet sizes, while descending the tree, the scheduler selects the child with minimum service received.

4.5.4 Recursive DAG Based Scheduling

It is also possible to traverse a directed acyclic graph (DAG) with a unique root (node with zero indegree) in a recursive round robin manner. At every step the scheduler starts with the root and descends down the DAG to reach a leaf (node with zero outdegree). Each node stores an index to keep track of the last visited decendent. Each time a node is visited, the decendent corresponding to its index is traversed and the index is incremented (modulo node's outdegree).

The service given to a node is equally divided among its decendents. The normalized service rate of root is one. This information can be propagated down the DAG to compute the normalized service rate of each node.

The properties of recursive DAG based scheduler can be derived along the same lines as that of binary RRR. The delay and fairness properties of a stream depend upon the number of leaves of DAG allocated to it.

4.5.5 Hierarchical Scheduling

The schedulers discussed so far schedule streams on an output link. It may be useful to schedule a set of input streams to get an output stream. There may be a number of such schedulers generating a number of output streams. These streams can in turn be scheduled by another scheduler on a link. This process can be repeated hierarchically over a number of levels. The motivation behind building such schedulers is to use simple schedulers like static priority, RRR etc. as basic building blocks so as to construct hierarchical schedulers which meet the desired requirements. This idea first appeared in the *rate control static priority* scheduler as described in [128] and was later refined and formalized for the fair queuing algorithms in [129]. work-conserving RRR has a bounded worst case fairness index (WFI). Therefore, the corresponding hierarchical scheduler approximates H-GPS. As a result, RRR can be directly applied to implement concepts such as link sharing.

Finally it is noted that the variants of RRR discussed above in this section share most of the good properties of the basic RRR algorithm outlined in the chapter, namely, simplicity, support for wide rate range, ease of hardware and software imple-

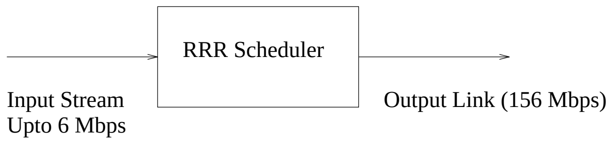

Figure 4.14: Simulation scenario.

mentation and possibility of distributed implementation. The proofs of fairness and delay bound can be easily extended to these variants.

4.6 Simulation Results

In this section, we present some preliminary simulation results for RRR. The simulation scenario consists of several input streams scheduled on same output link as shown in Figure 4.14. Input streams are chosen from a uniform distribution between 1 and 6Mbps (6Mbps is expected bit rate of MPEG-II streams). The streams are scheduled on a synchronous output link of 156Mbps. Fifteen bits were used to represent a rate, to ease the simulation on 32 bit computers. Note that the basic RRR algorithm provides perfect isolation. The delay of a cell of a stream is independent of arrival pattern of cells of other streams. Therefore the delay distribution of cells of a single stream will remain same irrespective of the cross traffic.

While constructing the scheduling tree, the phase at each level was assigned randomly. They were repeated a number of times with different seeds of random number generator to make sure that the results are consistent and the simulations are correct. Figure 4.15 shows the delay of successive cells of a stream for a part of a simulated period.

Figure 4.16 plots the statistical delay distribution of cells of the stream. The X-axis represents the delay of cells which varies from zero to the theoretical delay bound. The Y-axis plots the cumulative number of cells having delay in the corresponding range. It was found that with the random phase allocation, the delay bound was never reached. However with a specific phase allocation the tail of the delay distribution approached close to the theoretical delay bound.

The statistical delay distribution of cells of the stream was found to be a bell

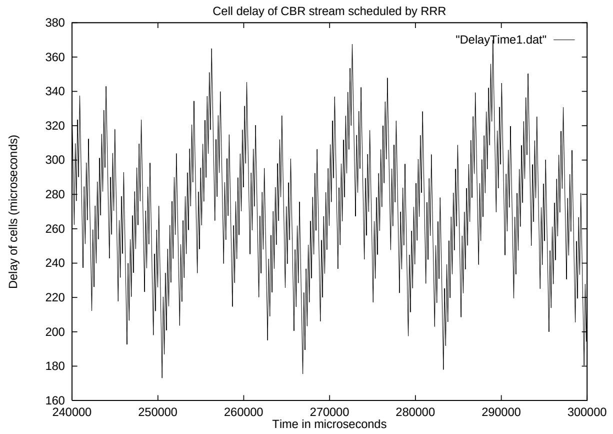

Figure 4.15: Delay of successive cells.

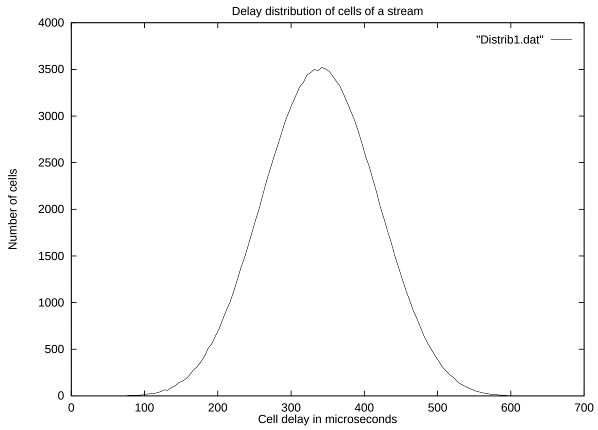

Figure 4.16: Distribution of delay of cells.

shaped curve, with maximum delay less than the delay bound. With the random phase allocation, the delay bound was never reached, but the simulation showed that the delay bounds were reasonably tight.

4.7 Implementation Considerations

It may be noted that the number of leaves in the tree is bounded by g times the number of streams being scheduled (N). Therefore the space requirement for the scheduler to store the scheduling tree is bounded by $2Ng$.

In order to schedule a cell, the scheduler descends down the scheduling tree to a leaf. The height (the maximum level of a node in the tree) of the tree is bounded by g. Thus, in the worst case, a naive implementation will take no more than g operations to schedule a single cell.

However, clever implementations can give better performance in terms of the time needed to schedule a cell. Given a binary RRR scheduling tree with any rate allocation for streams, it is straightforward to construct a corresponding 4-ary RRR scheduling tree by merging nodes at every odd level of the binary tree with their parent, such that the schedule generated by the binary and 4-ary schedulers on the corresponding tree is identical. The tree is constructed starting from the root node and collapsing two levels of the binary RRR tree into a single level of 4-ary tree. In case there is a leaf node in the first level of the binary RRR tree, two copies of the node are made in the 4-ary tree. The four children of the root node in the 4-ary scheduling tree are positioned so that they are visited in the same order as in the binary RRR tree. This process is continued till all the nodes (including leaves) of the binary RRR tree are collapsed into the 4-ary scheduling tree.

The height of 4-ary tree will be $\frac{g}{2}$. As a result, $\frac{g}{2}$ operations are needed in order to schedule one cell. This improvement in the number of operations is at the cost of a factor $\frac{4}{3}$ increase in space needed to store the 4-ary scheduling tree. The number of leaves in 4-ary scheduling tree are bounded by two times the number of leaves of corresponding binary tree i.e. $2Nc$. The number of internal nodes in the 4-ary tree will be $\frac{4}{3}2Nc$. There is a $\frac{4}{3}$ factor increase in storing space to reduce the number of operations in scheduling a cell by factor of two. For any software implementation, there is a tradeoff between time required to schedule a cell, amount of memory available to scheduler to maintain its data structures.

For the fixed point rate representation, both the space requirements and the number of operations to schedule a cell are dependent on the maximum number of streams to be scheduled (N), and the the rate granularity (g) is clear from the above expressions. Similarly for the floating point rate representation, the rate range (the maximum value the exponent of rate can take) will also be a factor in the dependence.

The basic unit of a counter based hardware implementation of the RRR scheduler is shown in Figure 4.17. There is one such unit for each leaf node in a binary scheduling

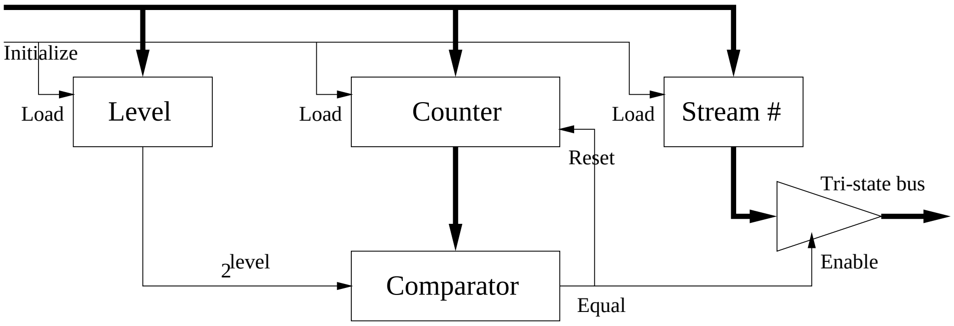

Figure 4.17: The basic unit of a suggested hardware implementation of RRR scheduler.

tree. The output of each such unit is connected to the same tri-state bus. Let E be the maximum range which the exponent of the normalized rate can take. The maximum depth of scheduling tree is bounded by $E + g$. The counter and comparator has a width of $E+g$. Every counter is initialized to $(2^{\text{level}} - \text{phase})$ of its corresponding leaf. The level of the leaf and its stream number are stored in corresponding registers upon initialization. All the counters are clocked by the same clock and are incremented every clock cycle. As soon as the counter reaches a value of 2^{level}, it is reset to 0 and the associated stream number is output to the tri-state bus. If the phases and level of each counter is initialized in accordance with the scheduling tree, then at each clock cycle, exactly one counter is guaranteed to reset and output its corresponding stream number at the tri-state bus. This output is exactly same as the schedule generated by the RRR scheduling tree. This can be used to select the appropriate output queue to copy cells to the output link.

Corresponding to the Ng leaves in a scheduling tree, there are Ng counters and the same number of associated registers. Each counter counts up to $2^{E+g}-1$. Each register will store the phase information of $E+g$ bits, the level information of $\log(E+g)$ bits and the stream number of $\log N$ bits.

The amount of logic needed for this implementation is $Ng(E+g)$ for the counters, $Ng(\log(E+g) + \log N)$ for the registers and $Ng(E+g)$ for the comparator. The speed of this hardware implementation depends upon the delay of the tri-state bus and the clock skew. The delay of the tri-state bus as well as the clock skew depends upon Ng. We believe that it is possible to implement such a hardware at very high clock speeds.

4.8 Conclusions

A simple and new scheduling algorithm called recursive round robin (RRR) has been described in this chapter. This scheduler can be used to schedule streams on a link with certain constraints. The scheduling algorithm guarantees a delay and jitter bound on compliant streams. Proofs of delay bound and jitter bound are provided for generic traffic descriptors. Several variants of the RRR scheduling algorithm, including the variable packet size scheduler are described. It has been shown that the work-conserving version of the RRR is fair. Bounds on two different fairness indices are analytically derived. Because of bounded worst case fairness index, it is argued that hierarchical scheduler derived from work-conserving RRR approximates H-GPS. It would be easy to generalize this result to other work-conserving variants of RRR. Proof of worst case delay bound remains valid for the work-conserving RRR. Implementation tradeoffs for hardware and software implementation have been discussed.

Proof of delay bound has been supplemented by simulations. The delay bound proven for the RRR scheduling algorithm is tight but there is a room for further improvement by finding *efficient node allocation strategies*. Another area that this work can further be extended is to study the detailed performance for those proposed variants of the basic RRR.

Because of its bounded delay, bounded buffer and bounded fairness properties, the RRR scheduling algorithm is suited for scheduling packets in real-time networks. Simplicity, low implementation complexity, and possibility of very high speed implementations, make this algorithm particularly suitable for ATM networks. Software implementation may be deployed in network interface cards whereas hardware implementation is well suited for high speed ATM switches and high end network interface cards.

Application of RRR algorithm in designing media access control (MAC) protocol for wireless ATM networks is an area of further work. It is possible to implement

RRR in distributed environment. Since RRR already provides guaranteed QoS and good fairness properties, a distributed implementation can act as a MAC protocol which in addition to high link utilization, also provides QoS guarantees to individual connections.

Chapter 5

Schedulers for Bounded Delay Service in Virtual Networks

Virtual networking is an important step in the evolution of data networks. It allows quick deployment of new services over legacy networks, eases network operation and management by hiding the unnecessary details and presenting a simplified topology, allows development of experimental protocols in a controlled and safe environment and eases interoperability between networks of different types.

Providing QoS guarantees to real-time applications in virtual networks is as important as in physical networks. For example, future corporate virtual private networks (VPN) will carry real-time voice and video conference data along with the regular application data. In order to successfully transport this voice, video and multimedia data, QoS guarantees must be provided to such traffic. Most of the research in virtual networks has been focused on developing new services or managing the virtual networks more efficiently. For instance, Virtual Private Networks [109] over the Internet are used by corporations for enhanced security. The Geoplex system [91] is an IP based service platform that offers support for rapid automated deployment and management of overlay networks. The ongoing work in the Genesis project [124] and the Netscript project [36] aim towards developing programmable virtual networks.

Traffic characterization and admission control functions needed for providing QoS guarantees are not dependent on lower layer functions of the network. Therefore they can be applied to virtual networks just as they are applied in physical networks. However the scheduling algorithms and their properties heavily depend on how the traffic is actually transported. Therefore, there is a need to reexamine the scheduling algorithms in the context of virtual networks.

The research related to scheduling algorithms for virtual networks has been limited. In [38, 129, 8, 9, 120] the concept of link sharing and packet scheduling algorithms to implement link sharing have been proposed. These algorithms hierarchically partition the link bandwidth into several classes while providing bounded delay and rate guarantees to traffic from each subclass. Application of link sharing to build QoS capable virtual networks, where a virtual link is realized by a sequence of logical links has not been discussed.

While providing QoS in a virtual network, it is desirable to maintain the partitioning introduced by the virtual network. The physical network should keep the state information only for the tunnels, not for individual sessions of virtual network. The entire traffic of a tunnel should be transported as if it was a single session of the physical network. We discuss this approach in this chapter. An alternative approach could be to tag each packet with the QoS required by it (like delay bound or priority) and schedule the packets individually at the physical network [21]. This approach is currently not practical as it requires significant per packet processing. The differentiated services working group [11] has proposed a scheme for tagging packets with their QoS. The scheme is still evolving and its extension to virtual networks is yet to be explored. Another approach could be to maintain, at every physical link, state information corresponding to each session of the virtual network passing through the link and manage sessions individually at physical network. This not only introduces scalability concerns, but also requires integration of virtual network signaling with that of the physical network. In addition, it requires that the schedulers at each node in the physical network distinguish between traffic of different sessions in the virtual network, which may not be possible because of traffic aggregation.

We first describe an abstract model of virtual networks in Section 5.1. Based on this model we show, in Section 5.2 that work-conserving schedulers can not provide bounded delay to sessions of virtual network. We show that if work-conserving schedulers are used at every place of contention in the network, then misbehaving sessions can affect QoS guarantees of well behaved sessions sharing the same virtual link. This problem arises when traffic at a rate larger than the allocated rate is sent on a virtual link. We quantify this excess traffic as output burstiness and show that output burstiness is an inherent lower bound on worst case session delays. In Section 5.3 we extend the theory of latency rate servers to show how latency rate servers with bounded output burstiness may be used to provide bounded delays to sessions of virtual network. This gives a flexible method for designing a generic class of scheduling algorithms which may be used in virtual networks. Because of the output burstiness constraint, the bandwidth sharing among real-time traffic is limited within the same virtual link. In Section 5.4 we briefly discuss how best-effort traffic can utilize the residual bandwidth to achieve more sharing. We also present an example of scheduler for virtual networks in Section 5.5. We summarize our results in Section 5.6.

5.1 Virtual Networks

A virtual network is a network overlaid on another physical network. The underlying physical network may have a completely different nature. It may be of large size and may have limited capabilities like poor security support. The physical network may be based on legacy systems which cannot be upgraded. It may even be running a completely different set of protocols. Overlaying a virtual network over a physical network helps in rapid creation and deployment of new services over the legacy network [124, 36, 91]. The virtual network may provide enhanced security [109], and better network management and control functions. Virtual networks may also be used to provide interoperability between networks of different types.

Some examples of virtual networks are an ATM network carrying virtual connections over virtual paths [16] (for simpler network operation and management), IP over ATM networks (for interoperability), virtual private networks in the Internet [109] and IPV6, and the MBONE and 6-bone virtual networks [30, 82] over the Internet (for deploying multicast services over the legacy network).

Figure 5.1 shows a virtual network overlaid on a physical network. It consists of

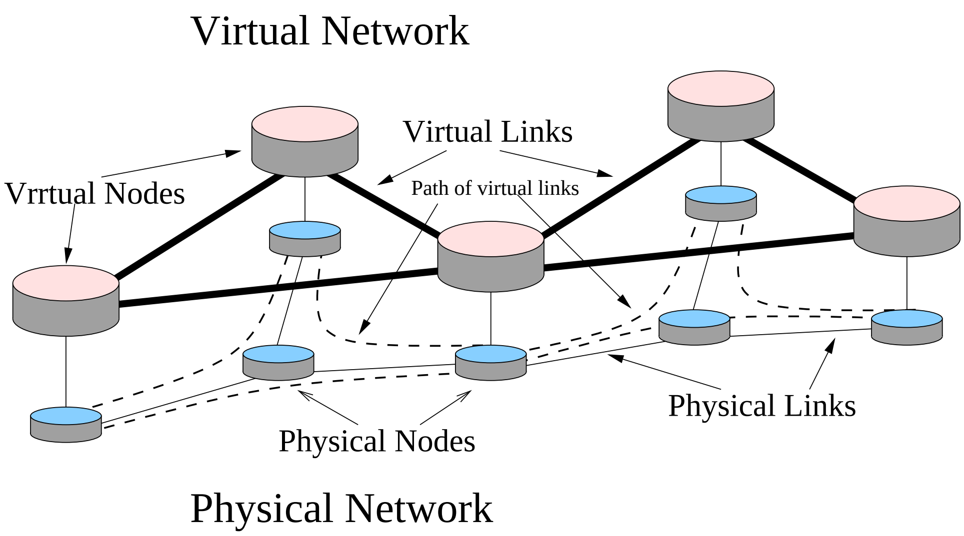

Figure 5.1: Example of a virtual network.

virtual nodes overlaid on the physical nodes. The virtual nodes are connected to each other by virtual links. A virtual link is either a direct link or a path between two nodes in the physical network. Packets arriving on a virtual node are forwarded to the next virtual node via a virtual link. This forwarding decision is made by looking up the packet header and routing table of the virtual network. In order to forward a packet on a virtual link, it is first encapsulated [57, 112] into one or more packets (a packet may have to be fragmented) of the physical network. These packets are tagged with the required header needed to route them to the destination physical node corresponding to the next virtual node and sent on the physical link. The packets pass through a series of physical nodes which lie in the path of the virtual link before arriving at the next virtual node. At each physical node, the packets are forwarded based on their header and routing tables in the physical network. At the destination node, the encapsulated packet of the virtual network is reconstructed and given to the virtual node.

Thus, the packet forwarding in a virtual network is done at two levels. At a higher level, packets are forwarded at virtual nodes to different virtual links. This is done by encapsulating and tagging them with the header specific to the virtual link to which they are forwarded. At lower level, packets are forwarded to different physical links based on their header and the routing tables of the physical network. Note that the initial paths of two different virtual links starting at the same virtual node may be the same (they may diverge only after traversing some common physical nodes). The physical network distinguishes between packets of different virtual links by the information present in their headers. These headers are created at the time of encapsulation and are specific to the virtual link on which the packets have been forwarded. Therefore packets of different sessions, when tunneled through the same virtual link, have the same headers, and the physical network gives the same treatment to all the traffic of a virtual link.

With this approach the complexities of the physical network as well as the virtual network are reduced. The physical network only needs to manage the aggregated traffic in the form of tunnels and the virtual networks are presented with a simplified topology, hiding the irrelevant details.

5.2 Drawback of Traditional Schedulers

Packet scheduling algorithms are key in providing QoS in any network. In most of the switches and routers, there is a scheduling algorithm at each output queue which decides the relative order in which packets from different sessions are sent on the link. By controlling this ordering, a scheduler can give rate and delay guarantees to individual sessions. There are a number of properties which a scheduler must satisfy. It must provide a local delay bound, i.e., a bound on the maximum delay incurred by packets of a session in the output queue. Similarly a scheduler must also bound the delay jitter and loss rate for the traffic. It should also provide a bounds on buffers needed at various queues in the network. In addition a scheduler must isolate one stream from other streams so that it is able to maintain the performance guarantees for such a stream even in the presence of misbehaving streams in the system. Finally, the scheduler should be able to divide the available link bandwidth among competing streams in a fair manner. This property is particularly important for the design of work-conserving schedulers.

In general, schedulers can be characterized as work-conserving or non-work-conserving.

A scheduler is said to be work-conserving if the scheduler is never idle when at least one packet is buffered in the system. A non-work-conserving scheduler may remain idle even if there are available packets to transmit.

Examples of work-conserving schedulers are first come first served, static priorities and the class of fair queueing schedulers like weighted fair queueing (WFQ) [86], virtual clock [131], self clocked fair queueing (SCFQ) [52] and recursive round robin (RRR) [44]. Non work-conserving schedulers include rate controlled static priority (RCSP) [130], delay earliest deadline first (Delay-EDF) [47] and jitter earliest deadline first (Jitter-EDF).

In a virtual network, the scheduling is done at two different levels. At the level of physical network, there is a scheduler, scheduling aggregate traffic of different virtual links. We call this scheduler as the link level scheduler. The link level schedulers treat the entire traffic of a virtual link as a single session. They do not distinguish between traffic of different sessions of virtual network passing through the virtual link. At the level of a virtual node, the packets from different sessions are queued before they can be encapsulated and sent on the virtual links. There is a scheduler which decides the relative ordering of packets from different session. We call this scheduler as the virtual scheduler. The virtual scheduler of a virtual link schedules packets only if the corresponding link level scheduler schedules a packet of its virtual link. This may be visualized as a hierarchical scheduler [129] with two levels of hierarchy. At the top level of the hierarchy, the link level scheduler partitions the link bandwidth into virtual links. At the next level, the virtual scheduler divides the bandwidth of the virtual link among the sessions. Note that a hierarchical scheduler is work-conserving if and only if all the schedulers in the hierarchy are also work-conserving.

In case hierarchical schedulers like those proposed in [8, 129, 120] are used in a virtual network, the complete link sharing hierarchy [38] needs to be maintained at each link. Therefore, the link level scheduler would require relevant information about every session of virtual network passing through the link. This destroys the partitioning introduced by virtual networks. The physical network needs to know about sessions of the virtual network, whereas the virtual network needs to know the topology of the physical network down to the link level details. However, using

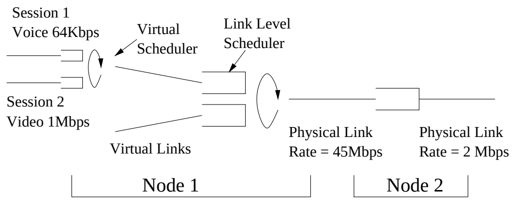

Figure 5.2: Queues at two different nodes in the path of a virtual link.

hierarchical schedulers only at virtual nodes poses another difficulty. All the schedulers presented for hierarchical link sharing are work-conserving as they attempt to redistribute the residual bandwidth on a link in a fair manner. We next show that if a virtual scheduler and the link level scheduler at a node in a virtual network, are both work-conserving, then it is impossible to provide bounded delay service in a virtual network (unless the packets themselves carry their delay information explicitly in their headers). As a result the schedulers presented in [8, 129] cannot be used in a virtual network. This calls for a design of scheduler which can be used in a virtual network.

The basic difference in a virtual network is due to the fact that the physical network only differentiates between the traffic of different virtual links. The packets of different sessions of virtual network, passing through the same virtual link are tunneled through the same path. If individual packets headers do not carry any information about their current delays, priorities or QoS required then the physical network cannot distinguish between packets of different sessions of virtual network which pass through the same virtual link. As a result it cannot give different scheduling priorities to such packets. All the traffic traversing a virtual link is serviced in first come first serve manner by the link level schedulers. Therefore, it is difficult to provide isolation between traffic of different sessions of the virtual network which passes through the same virtual link.

Let a virtual link of rate 1.5Mbps be realized by a two hop path in the physical network as shown in Figure 5.2. Let the first physical link in the path have a rate of 45Mbps and let the second physical link be of rate 2Mbps. Assume that one voice sessions of rate 64Kbps and a video session of rate 1Mbps are carried by this virtual link. Also assume that the link level scheduler and the virtual scheduler at the first node (node 1) are both work-conserving. Assume that no other traffic is present in the system. Assume that for some reason, the video session misbehaves, and queues up a large amount of data in its output queue. Since the virtual scheduler and the link level scheduler at the first node are both work-conserving, the entire traffic of the video session (along with a little traffic of voice session) will be sent from the first node at a rate of 45Mbps. The rate available to the virtual link at the second node (node 2) is limited by the physical link rate which is 2Mbps. This will result in large queues for the virtual link at node 2. The link level scheduler cannot distinguish between the traffic of voice session and video session. It simply serves all the virtual link traffic in first come first serve manner. This results in large delay at the second node for the voice traffic. Thus a session which is well behaved may get penalized because of a misbehaving session in the same virtual link, if the link level schedulers and virtual schedulers are both work-conserving.

Each virtual link is allocated a rate to carry its traffic. The problem arises when a work-conserving scheduler sends excessive traffic into a virtual link at a rate higher than the rate allocated to the virtual link, which gets queued up somewhere deep in the network. Since there is no differentiation of the traffic from different sessions after the traffic enters a virtual link, there may be large end to end delays even for the sessions at the virtual link which did not send any excess data. We define the term output burstiness to characterize the excess traffic sent on a virtual link.

Definition 5.2.1 (Output burstiness) *The output burstiness of a virtual link l is b_l, if for any interval of the form $(t_1, t_2]$ the following is true:*

$$S(t_1, t_2) \leq r_l(t_2 - t_1) + b_l$$

where $S(t_1, t_2)$ is the amount of traffic of virtual link scheduled by its virtual scheduler in the interval $(t_1, t_2]$ and r_l is the rate of the virtual link l.

The output burstiness of a virtual link is equal to the excess traffic sent by its virtual scheduler, at a rate greater than the virtual link rate, as shown in Figure 5.3.

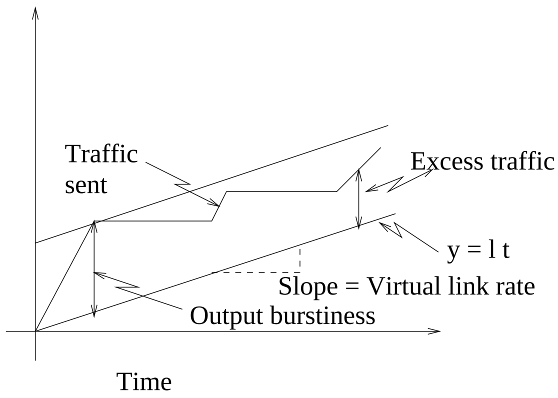

Figure 5.3: The output burstiness of a virtual link.

Bounded output burstiness provides an upper bound on the amount of the traffic which may be sent on a virtual link in a given time interval. Output burstiness is a property of a virtual link which depends upon its virtual scheduler and the link level scheduler at the first physical link in the path of the virtual link. It does not provide any stringent bound on the amount of traffic on individual sessions which may be sent on the virtual link.

An output burstiness of zero means that the rate of the traffic sent on a virtual link never exceeds the rate of the virtual link. This also implies that every packet will start a new session busy period at the next downstream node in the path of the virtual link.

Note the case where session delays are greater than or equal to the output burstiness. Even if the downstream scheduler is an ideal fluid-flow scheduler serving at a constant rate of r_l, it may take time equal to $\frac{b_l}{r_l}$ to clear its queue. A large value of output burstiness results in large delays at downstream nodes. It is therefore desirable to have the output burstiness as small as possible and independent of the number of sessions passing through the virtual link.

The hierarchy of the virtual scheduler and link level scheduler at the entry node of a virtual link may also be viewed as a single scheduler acting on the sessions of virtual network at that node. This gives us an option of using non hierarchical schedulers like EDF [47], RCSP [130] which are non-work-conserving. We now show that it is not possible to replace the hierarchy of the virtual scheduler and the link-level scheduler at virtual nodes by a non-hierarchical scheduler, while still providing low delay bounds to sessions of virtual network. We show that in case single level flat schedulers are used, the output burstiness of virtual links becomes proportional to the number of sessions passing through the virtual link. Since the worst case delays are greater than or equal to the output burstiness, the delay bound becomes proportional to the number of sessions passing through the virtual link.

Consider a physical link of one unit bandwidth partitioned into two virtual links of rate 0.5 units each. Let this physical link be the first link in the path of both the virtual links. Assume that there are N sessions passing through each of the virtual links. Let these sessions be numbered from 1 to $2N$ and let all of them be allocated the same bandwidth of $\frac{1}{2N}$. Any non-hierarchical scheduler will schedule packets from these sessions without taking into account the grouping of these sessions into the virtual links. Therefore the order in which packets are scheduled is independent of the virtual links to which these sessions belong.

Assume that each session sends constant size packets of size S each. Also assume that each session always has packets waiting to be served. Since the rate allocated to the sessions is exactly equal to the link rate, and the sessions always have data to send, the scheduler cannot remain idle for an unbounded amount of time. Let t be a time instant such that the scheduler is not idle for time more than S in the interval $[t, t+NS)$. Let the first $N-1$ packets served by the scheduler in this interval belong to sessions $c_1, c_2, \ldots, c_m, m \leq N-1$. Since the scheduler is non-hierarchical, the scheduling order will remain the same irrespective of the assignment of these sessions to virtual links. Let us assign the sessions $c_1, c_2, \ldots, c_m$ to virtual link l_1 and rest of the sessions to virtual link l_2. Note that the number of sessions on virtual link l_1 is at most $N-1$, therefore the rate allocated to virtual link l_1 is less than $\frac{N-1}{2N}$. Also note that the total traffic sent on this virtual link in interval $[t, t+NS)$ is at least $(N-1)S$. Therefore we have:

$$r_{l_1} \leq \frac{N-1}{2N} \tag{5.1}$$

$$S_{l_1}(t, t+NS) \geq (N-1)S \tag{5.2}$$

From the above two equations we have:

$$S_{l_1}(t, t+NS) - r_{l_1}NS \;\geq\; \frac{1}{2}(N-1)S \tag{5.3}$$

Therefore the output burstiness for virtual link l_1 is at least $\frac{1}{2}(N-1)S$, which is proportional to the number of sessions passing through the virtual link.

Thus a non hierarchical scheduler cannot replace the hierarchy of the virtual scheduler and the link level scheduler at virtual nodes, while providing low delay bounds to sessions.

5.3 Bounded Delay Service in Virtual Networks

The theory of latency rate servers is presented in [117] . This theory provides a general model to study the worst case behavior of individual sessions in a network of schedulers where the schedulers in the network may employ a broad range of scheduling algorithms. Using this theory, tight end to end delay bounds for an arbitrary network of schedulers can be proved. The theory of latency rate server provides a means to describe the worst case behavior of a broad range of scheduling algorithm in a simple and elegant manner. For a scheduling algorithm in this class, it is only required that the average rate of service of service offered by the scheduler to a busy session, over every interval starting at time θ from the beginning of the busy period, is at least equal to its reserved rate. A number of schedulers including weighted fair queueing (WFQ or PGPS), virtual clock, SCFQ, weighted round robin, deficit round robin, and recursive round robin belong to the class of latency rate servers.

An obvious approach to provide bounded delay service in a virtual network would be to apply the theory of latency rate servers to a virtual network. Unfortunately, this theory cannot be used in its present form in a virtual network. Even if all the schedulers in the virtual network belong to the class of latency rate servers, the worst case delays for individual sessions cannot be bounded. In a physical network, the schedulers at every node in the network satisfy a *latency rate* property for traffic of every session passing through the node. In a virtual network, traffic of multiple sessions is aggregated into a virtual link. The link level schedulers present at intermediate nodes in the path of virtual link satisfy latency rate property for aggregate traffic on the virtual link, but they do not satisfy the same property for individual sessions of the virtual link. The main reason for this is that the link level schedulers cannot provide isolation between different sessions of virtual network in the presence of large output burstiness.

We extend the concept of latency rate servers so that it is also applicable to virtual networks. We show that if output burstiness of a scheduler is bounded and it satisfies the latency rate property, then a tandem of such schedulers also satisfies the latency rate property, even if the downstream schedulers act on aggregate traffic. As a result, there is no need to satisfy the latency rate property for individual sessions in the path of a virtual link. In particular, if a virtual link passes through a series of link level schedulers such that: (a) each of the scheduler can be modeled as latency rate servers for the aggregate traffic on the virtual link, (b) the virtual scheduler at the beginning of the virtual link can also be modeled as latency rate server for individual sessions of the virtual network, and (c) the output burstiness of the virtual link is bounded, then the virtual link along with its virtual scheduler acts as a latency rate server for individual sessions. In a virtual network, a session passes through a series of virtual links. The latencies of the virtual schedulers of these virtual links can be combined to obtain tight end to end delay bounds for individual sessions (as in physical networks). Thus the theory of latency rate servers may be extended to the environment where at some nodes in the network, latency rate property is satisfied for aggregate traffic, not for individual sessions. As a result, this theory can be applied to a virtual network to carry out a tight analysis of end to end delay bounds and buffer bounds.

We first define latency rate servers and then discuss their properties. We then discuss the network model of a virtual network. We then show that a tandem of latency rate schedulers with bounded output burstiness in a virtual network is also a latency rate server. We analytically derive the latency of the equivalent server. We then show how this result can be used in designing virtual networks.

Before formally defining a latency rate server, we need the following definitions taken from [115].

Definition 5.3.1 (Session Busy Period) *A session i busy period is a maximal interval of time $(t_1, t_2]$ such that for any time $t \in (t_1, t_2]$, packets of session i arrive with rate greater than or equal to the allocated rate r_i of the session. If $A_i(t_1, t_2)$ is the amount of session i traffic arriving in the interval $(t_1, t_2]$ then*

$$A_i(t_1, t) \geq r_i(t - t_1), t \in (t_1, t_2) \tag{5.4}$$

Definition 5.3.2 (Session Backlog period) *A session backlog period at a server is the maximal interval during which the session has a positive backlog.*

Definition 5.3.3 (Latency Rate Server) *Let τ be the beginning of a busy period of a session and $S(\tau, t)$ be the amount of session traffic served by the server in the interval $(\tau, t]$. A server is in the class of latency rate servers if and only if for all times t after τ and until all the packets arriving in the busy period beginning at time τ are serviced,*

$$S(\tau, t) \geq r(t - \tau - \theta). \tag{5.5}$$

The parameter r is called the rate of the server and the parameter θ is called latency of the server.

Note that the latency rate server provides a lower bound on the rate at which traffic is served after the starting of a busy period. The average rate of service offered by any latency rate server to a session after a delay of θ from the beginning of a busy period is at least its allocated rate.

If a leaky bucket compliant session is served by a latency rate server of rate greater than or equal to the token rate of the bucket then the maximum delay experienced by any packet in the server is bounded.

Lemma 5.3.1 (Single Node Delay Bound) *If (σ, r') is the leaky bucket descriptor of a session served by a latency rate server of parameters (θ, r) such that $r \geq r'$, then the delay d of any packet in the server is bounded by the following equation*

$$d \leq \frac{\sigma}{r} + \theta \tag{5.6}$$

Lemma 5.3.2 (Network Delay Bound) *The maximum delay d of a session with leaky bucket descriptors (σ, r'), in a network of latency rate servers, consisting of K servers in series, is bounded as*

$$d \leq \frac{\sigma}{r} + \sum_{j=1}^{K} \theta_j \tag{5.7}$$

where θ_j is the latency of jth server in the network for the session and every server has a rate greater than or equal to r and $r \geq r'$.

Lemma 5.3.3 (Buffer Bound) *The maximum backlog $Q_k(t)$ in the kth node of a session is bounded as*

$$Q_k(t) \leq \sigma + r \sum_{j=1}^{k} \theta_j \tag{5.8}$$

where θ_j is the latency of the jth server and rate of each server is at least r.

Lemma 5.3.4 (Network of latency rate servers) *A network of K latency rate servers, with parameters (θ_j, r_j), connected in series is also a latency rate server with latency $\theta = \sum_{j=1}^{K} \theta_j$ and rate $r = \min_{j=1}^{K} r_j$.*

A number of scheduling disciplines including weighted fair queueing, have been shown in [117] to belong to the class of latency rate servers. However, we have shown that no work-conserving scheduler including weighted fair queueing may be used as virtual and link level scheduler to provide end to end delay bound to individual sessions. Thus the theory of latency rate servers is not directly applicable to virtual networks. A generic latency rate server cannot be used at all the places in the network. This is so because a latency rate server only bounds the minimum service rate which sessions are guaranteed to get. It doesn't provide any upper bound on the service rates. Thus a latency rate server may also send excess data on a virtual link (due to some misbehaving sessions) which gets queued up at downstream nodes in the network causing excessive delays to even well behaved sessions.

A tandem of latency rate servers in a physical network is also a latency rate server. But this may not be true in a virtual network because at link level aggregate traffic

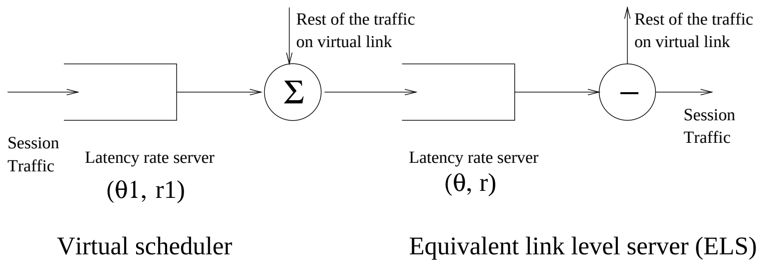

Figure 5.4: A logical model of a virtual network consisting of schedulers satisfying the latency rate property.

is scheduled. Therefore the link level schedulers may act as a latency rate server for virtual link traffic, but not for traffic of individual sessions of the virtual network. The logical model for a virtual network is shown in Figure 5.4.

The traffic of the virtual network is first scheduled by the virtual scheduler at virtual nodes. It is then scheduled by a series of link level schedulers in the physical network until it reaches the next virtual node. The virtual schedulers in the path of a session's traffic may be modeled as latency rate servers for individual sessions, but the link level schedulers at the physical nodes are latency rate servers only for the aggregate traffic on the virtual links, not for the individual sessions of the virtual network. We show that if the output burstiness of a virtual link is bounded then the tandem of latency rate servers as shown in Figure 5.4 is also a latency rate server with respect to the session traffic.

Consider a virtual link in the path of a session. Assume that the virtual scheduler of the virtual link can be modeled as a latency rate server of rate r_1 and latency θ_1. Let the rate of the virtual link be r_l. Also assume that the output burstiness of the virtual link is b_l. Let the aggregate traffic of the virtual link be served by a series of link level schedulers which are also latency rate servers. By Lemma 5.3.4 this series of schedulers can be modeled as a single latency rate server. Let the rate of this equivalent latency rate server be r ($r \geq r_l$) and its latency be θ. We claim that this tandem of two latency rate servers (shown in Figure 5.4) is also a latency rate server for the session traffic with rate r_1 and latency $\theta_1 + \theta + \frac{b_l}{r}$.

We begin by introducing some notation. Let $S_1^s(t_1, t_2)$ denote the amount of session traffic served by the first scheduler (virtual scheduler) in the interval $(t_1, t_2]$. Similarly let $S_2^s(t_1, t_2)$ denote the amount of session traffic served by the second server (equivalent link level server) in the interval $(t_1, t_2]$. Let $S_1(t_1, t_2)$ denote the amount of total traffic of the virtual link served by the virtual scheduler in the interval $(t_1, t_2]$. Similarly let $S_2(t_1, t_2)$ denote the total traffic of the virtual link served by the link level server in the interval $(t_1, t_2]$.

The session traffic served by the combination of the two servers is given by $S_2^s(.)$. In order to prove that the tandem of the two schedulers is also a latency rate server for session traffic, we need to provide a lower bound on $S_2^s(.)$ over a session busy period at the virtual scheduler. Following theorem summarizes our main result.

Theorem 5.3.1 *The amount of session traffic served by the combination of servers shown in Figure 5.4 at any instant t_2 in the session busy period starting at t_1, is bounded by:*

$$S_2^s(t_1, t_2) \geq r_1(t_2 - t_1 - \theta_1 - \theta - \frac{b_l}{r}) \tag{5.9}$$

At time t_2 there is some positive backlog of the session traffic in the system. This backlog is either divided between the virtual scheduler and the link level schedulers or the entire backlog is concentrated on only one of the schedulers. In case the link level scheduler has no backlog of session traffic, we may claim that the session traffic served by the virtual scheduler in the interval $[t_1, t_2]$ has also been served by the link level scheduler. Therefore in this case:

$$S_2^s(t_1, t_2) = S_1^s(t_1, t_2) \tag{5.10}$$

Since virtual scheduler is a latency rate server and t_1 marks the beginning of a session busy period at virtual scheduler, we have:

$$S_1^s(t_1, t_2) \geq r_1(t_2 - t_1 - \theta_1) \tag{5.11}$$

After substituting $S_1^s(t_1, t_2)$ from Eq. 5.10, the above equation may be rewritten as Eq. 5.9.

Now we need to establish Eq. 5.9 in case the link level schedulers have some backlog at time t_2. Assume that the last session busy period before t_2 at link level scheduler started at time t'. We look at the last packet served on or before t_2 at the link level scheduler. Let the arrival time of this packet at the link level scheduler be t''. We claim that:

$$t'' \geq t_2 - \theta - \frac{b_l}{r} \tag{5.12}$$

We prove this by contradiction. Assume that

$$t'' < t_2 - \theta - \frac{b_l}{r} \tag{5.13}$$

Since the current session busy period started at time t' we have $t'' \geq t'$. From the latency rate property of the equivalent link level server we have:

$$S_2(t', t_2) \geq r(t_2 - t' - \theta) \tag{5.14}$$

Since all of the above traffic arrived in the interval $(t', t'']$ from the virtual scheduler we have:

$$\begin{aligned} S_1(t', t'') &= S_2(t', t_2) && (5.15)\\ &\geq r(t_2 - t' - \theta) && (5.16) \end{aligned}$$

Substituting t_2 from Eq. 5.13 to get:

$$\begin{aligned} S_1(t', t'') &> r((t'' + \theta + \frac{b_l}{r}) - t' - \theta) && (5.17)\\ &= r(t'' - t') + b_l && (5.18)\\ &\geq r_l(t'' - t') + b_l && (5.19) \end{aligned}$$

This implies that the virtual scheduler violates the output burstiness constraint for the virtual link, which is impossible. Hence:

$$t'' \geq t_2 - \theta - \frac{b_l}{r} \tag{5.20}$$

The link level scheduler serves all the traffic on a virtual link in first come first serve manner. So, if it services all its traffic arriving till the time t'', it also services all the session traffic arriving till time t''. Thus we have:

$$\begin{aligned} S_2(t_1, t_2) &= S_1(t, t'') && (5.21)\\ \Rightarrow S_2^s(t_1, t_2) &= S_1^s(t_1, t'') && (5.22) \end{aligned}$$

From Eq. 5.11, substituting the value of $S_1^s(t_1, t'')$ in the above equation, we get:

$$S_2^s(t_1, t_2) \geq r_1(t'' - t_1 - \theta_1) \tag{5.23}$$

Substituting the value of t'' from Eq. 5.20 we have:

$$S_2^s(t_1, t_2) \geq r_1(t_2 - t_1 - \theta_1 - \theta - \frac{b_l}{r}) \tag{5.24}$$

This shows that the tandem of two latency rate servers as discussed above is also a latency rate server with rate r and latency $\theta_1 + \theta + \frac{b_l}{r}$.

With the above property we conclude that a virtual link which has a bounded output burstiness acts as a latency rate server for individual sessions even if the link level schedulers in the path of the virtual link do not provide any guarantee to individual sessions. With the help of this property, the entire theory of latency rate servers may be used in designing bounded delay service in a virtual network. We now show how to design virtual schedulers which have bounded output burstiness and satisfy the latency rate property for individual sessions.

5.4 Best-Effort Service in Virtual Networks

The output burstiness constraint limits the rate at which traffic may be sent on a virtual link. Sending excess traffic may result in large queues at downstream nodes in the path of the virtual link causing large delays to the traffic of the virtual link. This reduces the extent of statistical multiplexing which could have been achieved if the entire network was a single layered physical network. The amount of statistical multiplexing is limited among the traffic going through the same virtual link. The output burstiness constraint limits the aggregate virtual link traffic served by the scheduler in a given time, it does not limit the individual session traffic serviced by the scheduler. This means that active sessions may make use of the excess bandwidth available on a virtual link because of unallocated bandwidth or inactive sessions on the same virtual link. However, real-time sessions may not be able to use excess bandwidth on other virtual links even if these virtual links are carried on the same physical link.

Based on this observation we may classify schedulers as work-conserving or non work-conserving, with respect to a virtual link. We define a virtual scheduler of a virtual link to be work-conserving, if it permits the sharing of excess bandwidth on the virtual link. Formally, a virtual scheduler is virtual link is work-conserving if there exists a constant z such that in any backlog period (t_1, t_2) of the virtual link, the amount of traffic serviced is bounded below by

$$S(t_1, t_2) \geq r(t_2 - t_1) - z \tag{5.25}$$

This ensures that in presence of backlog, the scheduler will never be idle for a period more than $\frac{z}{r}$. For a work-conserving scheduler on a physical link, $z = 0$. In the case of a virtual link, there may be some irregularity in scheduling the virtual link traffic because of discrete packet boundaries resulting in a non-zero value of z.

The bandwidth sharing among different virtual links may be achieved by the best-effort traffic. Since there are no tight delay constraints on best-effort traffic, the excess from this traffic may be sent on a virtual link to utilize the available bandwidth on the physical link. At each virtual scheduler a separate queue for best-effort traffic is maintained. At link level schedulers, a separate best-effort queue is maintained for all the virtual links. The queues containing real-time traffic are given higher priority over the best-effort queues. Traffic is scheduled from best-effort queues if real-time traffic cannot be scheduled. Any work-conserving fair packet scheduler like weighted fair queueing may be used at the link level to ensure fairness to the non real-time traffic from different virtual links.

The link level schedulers need to distinguish the best-effort traffic from the real-time traffic of a virtual link. So there is a need to separate the two type of traffic. The separation between best-effort and real-time traffic of a virtual link may either be obtained using one bit marking (like CLP in case of ATM) in the packet header or by using different tunnels (using different virtual paths in case of ATM) for real-time and best-effort traffic of a virtual link.

There is an an end to end congestion control algorithm for each best-effort session which controls the rate at which the traffic is sent. This algorithm adjusts the sending rate to utilize the excess bandwidth available in the path of the session. Typical end

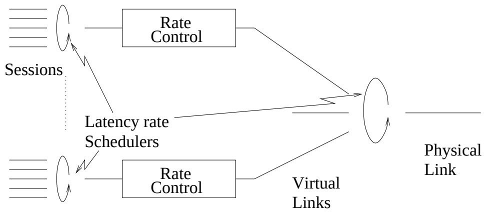

Figure 5.5: An generic latency rate scheduler with bounded output burstiness.

to end congestion control algorithms do not require the nodes in the path to explicitly maintain any state corresponding to each session. Therefore these algorithms continue to work well in virtual networks.

5.5 Example Schedulers for Virtual Networks

Figure 5.5 shows a generic scheduler with bounded output burstiness. The virtual scheduler for each virtual link is a latency rate server like WFQ. There is a rate control after each virtual scheduler which introduces inter-packet spacing equal to the packet size divided by the rate of the virtual link. Because of this rate control, the virtual link appears just like a physical link to the virtual schedulers. Because of discrete service boundaries, the rate control introduces an output burstiness of $\frac{S_{max}}{r_l}$ where S_{max} is the size of the maximum packet on the virtual link and r_l is the rate of the virtual link. After the rate control, the packets are queued before the link level scheduler, which is again a latency rate server like WFQ. There is an increase in end to end delay bounds of a session by S_{max}/r_l for each virtual link l that it passes through, as compared to a general hierarchical scheduler. This increase is expected to be small as compared to the latency of a single scheduler which is of the order of S_{max}/r_i, where r_i is the rate allocated to a session. In a typical situation, a virtual link will carry thousands of sessions ($r_l \approx 1000r_i$). Therefore the increase in the delay bound is expected to be less than 0.1 percent. On the other hand, with this arrangement, the virtual link traffic may be carried by the physical network just like ordinary sessions saving a lot of book keeping at the physical network.

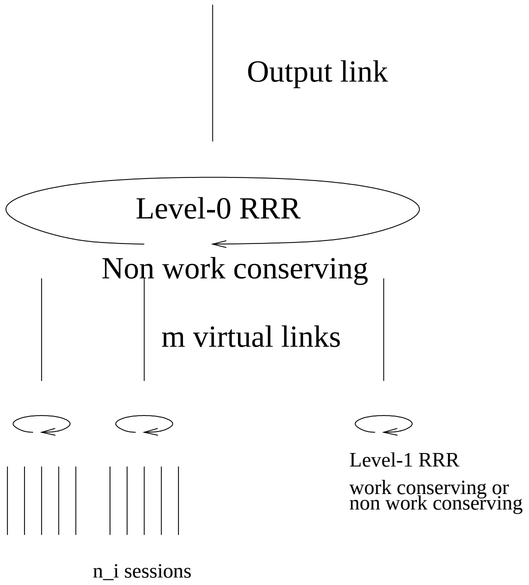

Figure 5.6: A two level hierarchical RRR arrangement for virtual networks.

5.5.1 Hierarchical RRR scheduler

In the above arrangement, the queues need to be maintained at the virtual scheduler as well as the corresponding link level scheduler. In addition, the virtual scheduler needs to use asynchronous timers to implement the rate control. By using a hierarchical RRR scheduler, it is possible to have a simpler implementation. In the rest of this chapter, we assume binary RRR scheduler for fixed packet size. The other variants may also be used in the same manner.

Assume that a virtual scheduler schedules sessions on m different virtual links which are labelled from 1 to m. Assume that virtual link i has n_i sessions passing through it. The virtual scheduler consists of one instance of level-0 RRR scheduler and m instances of level-1 RRR schedulers as shown in Figure 5.6.

The level-0 RRR scheduler partitions the link bandwidth among the m virtual links. This scheduler is non work-conserving which results in a bounded output burstiness for each virtual link. There are m level-1 RRR schedulers which schedule the traffic from different sessions. These schedulers may be work-conserving or non work-conserving.

With this arrangement, the leaves of the level-0 scheduling tree now point to

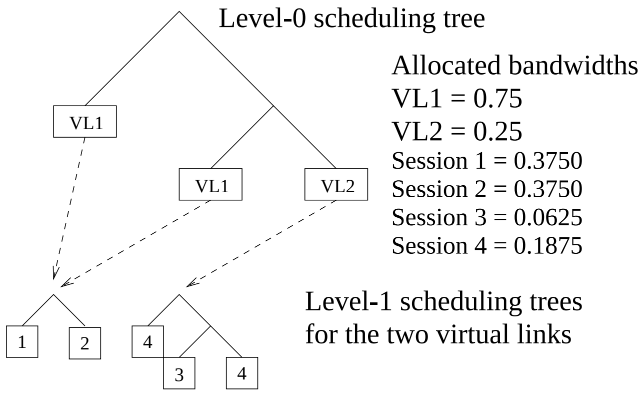

Figure 5.7: A sample scheduling tree arrangement for hierarchical RRR in a virtual network.

a level-1 scheduling tree corresponding to the virtual link represented by the leaf. While traversing the level-0 scheduling tree, when the level-0 RRR scheduler reaches a leaf. It traverses the corresponding level-1 scheduling tree and upon reaching its leaf, schedules traffic from the corresponding session. If the level-1 scheduler is work-conserving and the scheduler reaches a unallocated leaf or the session corresponding to the leaf has no data, the level-1 scheduling tree is traversed again starting from its root to go to the next stream. Figure 5.7 shows a sample scheduling tree arrangement for two virtual links.

Let the session with rate r_i be scheduled on a virtual link of rate r using a two level RRR scheduler. Let the physical link have a rate of l. Assume that c_i is the count of the number of ones in the binary representation of r_i/r and c is the count of number of ones in the binary representation of r/l. We have the following properties of this scheduler:

Lemma 5.5.1 *The output burstiness of the virtual link scheduled by the hierarchical RRR scheduler is cS_{max}, where c is the count of number if ones in the normalized binary representation of the virtual link rate.*

The top level scheduler is non work-conserving RRR. If $S(0, t)$ is the amount of traffic of the virtual link serviced in the interval $(0, t)$ then from simplification of Equation 4.1

we get:

$$S(0,t) \leq rt + cS_{max} \tag{5.26}$$

where r is the rate allocated to the virtual link and c is the count of the number of ones in the binary representation of the normalized virtual link rate. Since the choice of origin is arbitrary in the above equation, we may rewrite it as:

$$S(t_1,t_2) \leq r(t_2 - t_1) + cS_{max} \tag{5.27}$$

This shows that the output burstiness of the virtual link is bounded by cS_{max}.

Lemma 5.5.2 *The two level RRR acts as a latency rate server for the session with rate r_i and latency $(c_i/r_i + c_l/r_l)S_{max}$, where r_l is the rate of the virtual link, c_l is the count of the number of ones in the normalized rate of the virtual link and c_i is the count of number of ones in the normalized rate of the session.*

It has been shown in [129] that the worst case fairness index (WFI) of hierarchical scheduler is sum of the individual WFI of the schedulers in the hierarchy. The WFI of level-0 RRR is $\frac{c_l}{r_l}S_{max}$ and the WFI for level-1 RRR is $\frac{c_i}{r_i}S_{max}$. Therefore the WFI for the two level scheduler is $(\frac{c_i}{r_i} + \frac{c_l}{r_l})S_{max}$. The WFI of a scheduler is also an upper bound on its latency. Therefore the two level RRR is a latency rate server with latency $(\frac{c_i}{r_i} + \frac{c_l}{r_l})S_{max}$.

5.5.2 Flat RRR with virtual link based node allocation

We have shown in Section 5.2 that if a scheduler does not take into account the bandwidth partitioning induced because of virtual links, then the output burstiness of a virtual link can become very large (proportional to the number of connections passing through the virtual link). In this section we show that with a careful node allocation strategy in the scheduling tree of the RRR scheduler, the output burstiness of each virtual link can be bounded by a small constant. In this node allocation strategy the rates are first allocated to virtual links and then from the virtual links, the rate are allocated to individual streams. We call this allocation strategy as virtual link based node allocation. As a result, RRR with virtual link node allocation strategy is suitable for a virtual network. This results in lower latency as compared to the hierarchical RRR.

Recall that in the RRR scheduler, a (binary) scheduling tree is constructed which represents the rate allocation to various streams. The scheduling algorithm generates its schedule by traversing the scheduling tree in a recursive round robin order. The scheduling tree is modified only when new connections are added or old connections are removed. The leaves of the tree are labelled with stream numbers. We say that a leaf has been allocated to a stream if it is labelled with its stream number. This scheduling tree has the property called unfragmented rate allocation. According to this property, two nodes of the tree at the same depth can never to allocated to the same stream. This property is crucial for delay, buffer and fairness properties of the scheduler. However unfragmented rate allocation of the null stream (or the unallocated nodes) is not essential for these properties. Unfragmented rate allocation of the null stream is only needed for bounding the size of the scheduling tree.

In the virtual link based node allocation strategy, we relax the unfragmented rate allocation restriction for null stream. This gives more flexibility in node allocation which helps in reducing the output burstiness. Each stream is identified by a tuple (virtual link number, stream number). The stream number is defined in the context of its virtual link. Following the conventions of the RRR scheduling tree, the virtual link numbered 0 corresponds to the unallocated bandwidth on the physical link. Similarly, the stream numbered 0 of every virtual link corresponds to the unallocated bandwidth at the virtual link.

Construction of the scheduling tree

The scheduling tree is constructed in two phases. In the first phase, the rate of each virtual link is represented relative to the link rate and a scheduling tree is constructed as in normal RRR scheduler. This scheduling tree has unfragmented rate allocation for all the virtual links including the virtual link numbered 0. This scheduling tree is constructed at the time of creation of virtual links and is rarely modified. This tree needs to be modified only if the bandwidth allocation to the virtual links are changed. In the second phase, the allocation to individual streams is done. The rates of individual streams are represented relative to the rate of the physical link. The rate allocation for the streams is carried out as if it is being done on the physical link, with a small restriction. During this allocation if a stream is being added or deleted on virtual link numbered j, then only the nodes labelled with virtual link j are manipulated. The operations needed for addition or deletion of streams are *allocate, split, deallocate, join* and *phase exchange*. We now discuss how to carry out these operations in the second phase.

- Allocate: In normal RRR scheduling tree, a free leaf is allocated to the stream by writing the label of the stream number on the leaf. In the virtual link based allocation, if stream i is added to virtual link j, then only free leaves of virtual link j (labelled $(j, 0)$) can be allocated to stream i by relabelling them to (j, i).

- Split: In normal RRR scheduling tree, if there is no free leaf at the desired depth, then a free leaf at a smaller depth is split recursively until free leaves at the desired depth are created. In the virtual link based allocation, if stream i is added to virtual link j and a split operation is needed, only free leaves of virtual link j (labelled $(j, 0)$) are split into two free leaves of the same virtual link at one greater depth.

- Deallocate: In normal RRR scheduling tree, the leaf which is deallocated is simply labelled 0. In virtual link based allocation, the stream number of the leaf to be deallocated is changed to 0, the virtual link number of the leaf remains the same.

- Join: In normal RRR scheduling tree, two free sibling nodes are joined to give rise to one sibling at one lower depth. In virtual link based allocation, only two free sibling nodes of same virtual link may be joined to give rise to one free sibling of the same virtual link at one lower depth.

- Phase Exchange: In normal RRR scheduling tree, one free leaf at a level may be swapped (with a careful relabelling in a particular order) with another node at the same level. In virtual link based allocation, this swapping can be performed only if the free leaf and the node belong to the same virtual link.

With the above set of operations, it can be shown by using exactly the same argument as in normal RRR that if the available rate at a virtual link is greater than or equal to the rate of a new stream, it can be added in the virtual link. Similarly, it can be shown that all the streams (including the null stream) of every virtual link has unfragmented rate allocation.

Note that the set of unallocated leaves may not be unfragmented as there may be two unallocated leaves of different virtual links at the same depth. These leaves cannot be combined because the join and phase exchange operations can only be performed between nodes of the same virtual link. However, the unallocated leaves of the same virtual link are indeed unfragmented.

If there are m virtual links and the virtual link j has n_j streams then the number of leaves allocated to the virtual link j is bounded by $(n_j + 1)g$, where g is the maximum depth of the scheduling tree. Therefore the number of leaves of scheduling tree is bounded by $(\sum_{j=1}^{m} n_j + m + 1)g$. There are three terms in this expression. The term $(\sum_{j=1}^{m} n_j)g$ corresponds to the leaves allocated to the streams. The term mg corresponds to the free leaves in m virtual links and the term g corresponds to the free bandwidth at the physical link. The increase in size of the tree because of fragmented rate allocation of the null stream is only mg. This is just a 0.1 percent increase if there are about thousand sessions per virtual link.

The schedule generated by the recursive round robin traversal of the scheduling tree constructed above satisfies the delay bound and buffer bound properties for individual sessions. Therefore it is a latency rate server for traffic of session i with latency $\frac{1}{r_i} c_i S_{max}$ and rate r_i, where r_i is the rate of session i and c_i is the count of number of ones in the normalized rate representation of session i with respect to the rate of the physical link.

In addition to this, the tree has the following interesting property. If all the sibling leaves which belong to the same virtual link are joined until no two siblings can be joined, the resulting tree has at most c leaves per virtual link. This tree is identical to the tree constructed in the first phase of the rate allocation. The output schedule of these two trees is identical if individual stream numbers are ignored and only virtual link numbers of the output schedule are considered. As a result of this property, each virtual link has bounded output burstiness. If $S_l(0,t)$ is the amount of traffic of a virtual link l serviced in the interval $(0,t)$ then

$$S_l(0,t) \leq r_l t + c_l S_{max} \tag{5.28}$$

where r_l is the rate allocated to the virtual link and c_l is the count of the number of ones in the binary representation of the normalized virtual link rate. Since the choice of origin is arbitrary in the above equation, we may rewrite it as:

$$S_l(t_1,t_2) \leq r_l(t_2-t_1) + c_l S_{max} \tag{5.29}$$

This shows that the output burstiness of the virtual link is bounded by $c_l S_{max}$.

5.6 Conclusions and Future Work

We have shown that traditional work-conserving schedulers including the hierarchical schedulers are inappropriate for providing bounded delay service in generic virtual networks. The traffic of multiple sessions of a virtual network is tunneled through the physical network which treats the entire traffic of the tunnel as if it belongs to a single session. As a result, the physical network is not able to protect the well behaved sessions from misbehaving session which share the same virtual link. The problem can be solved by restricting the rate at which traffic may be sent on a virtual link. We introduce a term called output burstiness which we use to extend the theory of latency rate servers. We show that if a latency rate server also has a bounded output burstiness for each virtual link, it may be used in a virtual network to provide end to end delay bounds for sessions of the virtual network. This gives us a generic method to design a class of schedulers which may be used in virtual networks.

We have shown how latency rate schedulers used in ordinary networks can be modified for virtual networks by adding an output rate control. We proposed two versions of the recursive round robin scheduler for virtual networks. The hierarchical

RRR allows the sharing of residual bandwidth in a virtual link. The flat RRR with virtual link allocation does not allow sharing, but results in smaller latency and hence smaller delay bounds for sessions.

We show that real-time sessions of the virtual networks cannot use the residual bandwidth at other virtual links, sharing the same physical link. However, this bandwidth may be utilized by the best-effort traffic. In order to achieve bandwidth sharing by the best-effort traffic, the physical network needs to distinguish the best-effort traffic of a virtual link from its real-time traffic. This may be done either by creating separate tunnels for best-effort and real-time traffic or by marking the best-effort traffic if it is sent over the same tunnel. Further work is needed in this direction to explore the tradeoffs. Some congestion control algorithms require the participation of intermediate nodes in the path of the sessions. In some of these algorithms, the intermediate nodes only need to propagate congestion information to the sources, while in others the intermediate nodes need to carry out operations on a per session basis (like per session queueing, buffer management and scheduling). In a virtual network, per session operations may only be carried out on virtual nodes. The physical nodes can only carry out operations on the aggregate traffic. The performance study of these algorithms in a virtual network is an area of further study. It would also be interesting to come up with schemes for adapting these algorithms for virtual networks.

Chapter 6

Fair Sharing in Virtual Networks

6.1 Introduction

In this chapter we address the issue of sharing of residual bandwidth on physical links which carry multiple virtual links. In the last chapter we showed that virtual links of an Integrated Services virtual network cannot carry traffic at a rate higher than their allocated capacity even if the underlying physical links have large spare capacity to carry the traffic. Sending traffic at a rate higher than the allocated capacity of the virtual link results in large output burstiness which deteriorates the QoS properties of delay sensitive traffic on the virtual link. This results in wastage of unused capacity in the physical network. As a partial solution, to improve sharing we suggested using separate virtual links for real-time and best-effort traffic. Thus the best-effort traffic, which is not too sensitive to delays, may make use of the unused bandwidth on the physical links. However even with this, it is not possible to handle the best-effort sessions that need a minimum bandwidth guarantee. Some bandwidth may be reserved even for the best-effort tunnels, and the best-effort session requiring a minimum bandwidth guarantee may be admission controlled. This introduces additional undesirable partitioning of the virtual link capacity into best-effort and real-time classes and the sharing is still limited only to the "elastic" portion of best-effort traffic only. Therefore an integrated approach is needed which can extend residual capacity sharing even to real-time traffic and handle the best-effort traffic in the same framework.

In this chapter, we propose a new approach called Stochastic Fair Sharing (SFS) to improve sharing among virtual links of virtual networks. SFS resizes the capacity of virtual links, depending upon their traffic demand and residual capacity in the underlying physical network. The underlying scheduling weights are adjusted according to resized capacities. The capacity resizing is done at a granularity of session arrivals and therefore the resized capacity is available for session holding time. As a result, even the real-time traffic can make use of the increased capacity. While resizing redistributes the free capacity at the time-scale of session holding times, the packet level solutions (fair schedulers like WFQ, RRR) take care of instantaneous free capacities at the time-scale of packet transmission times.

This kind of approach is shown to result in factor 1.5 to 3 sharing gain in [27]. Similarly in [53] a pipe resizing approach for AT&T switched network is shown to result in a sharing gain of 2.

An important issue, which has not been discussed in any of these approaches is protection and fairness. In most of the cases every virtual link will have a provisioned capacity allocated at the time of establishment of the link. In case of virtual private networks (VPN), this provisioned capacity may also be included as a part of service level agreement (SLA) between the VPN customer and the service provider. It is possible that one virtual link resize its capacity to a large amount, whereas other virtual links may not be able to resize even to their provisioned capacities. According to the protection criteria, every virtual link should be able to resize its capacity to its provisioned value (within a reasonable time), even if its current capacity is lower than the provisioned value. This ensures that well behaved virtual links are not affected by misbehaving virtual links (those who asked for low provisioned capacity and then try to resize it to a large value) even in extreme conditions. Protection enables the lightly loaded virtual links to release their capacity for redistribution by the network. The virtual links may release their residual capacity only if they can get it back when they need it. According to the fairness criteria, the residual link capacities should be redistributed to virtual links in proportion to their provisioned capacities. In case of a network, this criteria is same as weighted max-min fair [64] allocation.

SFS ensures protection by allowing fair sharing of residual capacities. In case of a single link, the proposed SFS technique extends the admission control algorithm at the link and decides which sessions to accept and which to reject. For simplicity of exposition, we assume that QoS requirement of a session is expressed using a single bandwidth parameter. For VBR traffic the concept of equivalent bandwidth [54] may be used, for delay sensitive traffic the delay parameter may be translated to a bandwidth parameter (assuming that rate allocating schedulers [117] are used at this link), and for best-effort traffic the minimum bandwidth requirement of the session may be used. In order to make a decision about a new session, the classes are first sorted in increasing order of their normalized usage (current reservation divided by allocated capacity). A new session of a class is accepted only if the free capacity after accepting the session is greater than or equal to the sum of the trunk reservations of classes with lower normalized usage. Once a session is accepted, the corresponding weights in the underlying hierarchical fair packet scheduler (if any) are adjusted to reflect the change. In this way, this scheme attempts to equalize the normalized usage of classes by giving higher priority to classes with low normalized usage. Thus, SFS accounts for long term fluctuations in usage by carrying out capacity redistribution over a time-scale of session holding times, while the underlying hierarchical packet fair queueing algorithms handle short term fluctuations in traffic load by redistributing free capacities over the scale of packet transmission times. SFS ensures that while the free capacity of a class has been redistributed to other classes, it has low session blocking probability and can quickly regain its share of capacity as its session arrival rate increases.

In case of multi-hop virtual links of virtual networks, we assume that every virtual link has a provisioned capacity allocated by the service provider on its path at the time of establishment of the virtual link. However, depending upon its current and estimated load, a customer may request a change in the current capacity of its virtual link by sending an *increase* or *decrease* request in the provider's network. The increase request is admission controlled using SFS at each link of the provider's network. After processing these requests, the corresponding scheduling weights in the provider network are adjusted accordingly. In case of overload, the provider network generates *reduce* messages to request down-sizing of over-allocated virtual link capacities.

With this scheme we show that, the unbottlenecked virtual links achieve throughput equal to their current demand, and the equivalent capacity of 94% of bottlenecked virtual links is within $90 - 105\%$ of their weighted max-min fair capacity (weighted by provisioned capacity).

SFS can be used by service providers to ensure that bandwidth distribution is fair even in case of high and uneven load. SFS provides fairness by ensuring that every virtual link can increase its capacity up to its fair share within an interval of two mean session holding times. This also ensures that well behaving customers are protected from misbehaving ones (those who asked for low capacity paths and resize it to higher capacities).

A similar scheme to carry out *virtual partitioning* of a single link has been proposed in [12]. This scheme categorizes each class as underloaded or overloaded. The underloaded classes are given priority over the overloaded classes using the technique of trunk reservation. While this scheme ensures that the residual capacity of a class may be utilized by other classes, it does not attempt to do a fair redistribution of this free capacity. While using virtual partitioning, a class having a high session arrival rate may take the residual capacity of all the classes. The hierarchical virtual partitioning [93] suffers from the same drawback and is not applicable to multi-hop virtual links.

The proposed SFS scheme is simple, robust and can be implemented in large networks without significant overheads. In our simulations, the bandwidth penalty for using SFS was less than 2% and its average signaling load was less than 100 messages per second per router.

SFS may be used in telecommunication networks to achieve better sharing and lower call blocking probabilities. It may be used in the context of ATM networks to dynamically carry out fair bandwidth allocation to virtual paths [96]. Use of SFS to carry out dynamic bandwidth allocation to virtual links in a virtual network may result in better network utilization and fair resource sharing. In the context of differentiated services [11] in IP based networks, SFS may be integrated with the proposed bandwidth broker architecture to achieve fair sharing of portions of the network among several administrative domains.

This chapter is organized as follows. We describe our model in Section 6.2. SFS admission control procedure is described in Section 6.3. We discuss the case of multi-hop virtual links in Section 6.4. An approximate model to determine the trunk reservation parameter is presented in Section 6.5. Single link simulations of SFS are presented in Section 6.6 and simulations of SFS on a network are presented in Section 6.7. We describe future directions in Section 6.8 and conclude in Section 6.9.

6.2 Model

We first describe the single link case. We assume that a link of capacity C is to be partitioned into N logical links (or classes) of capacities C_i, such that $\sum_{i=1}^{N} C_i \leq C$. We assume that real-time sessions with bandwidth requirement arrive randomly as a stochastic process. Bandwidth is reserved upon session arrivals and is released upon session completion (using a protocol like RSVP [132], PNNI [2] or OpeNet [19]).

We work with simplistic assumptions of Poisson session arrivals and exponential session holding times. In a typical scenario, these logical links will belong to different networks (of different organizations). We therefore assume that the session arrivals at these links are independent of each other.

There is an admission control entity at the link which decides whether the link has adequate free capacity to accept the reservation requests of sessions. In case the reservation request of a session can't be accepted, we say that the session is blocked. An important performance parameter, in the context of telecommunications networks is the session blocking probability. In these networks, the link capacities are provisioned to keep the call blocking probabilities below a specified value (say 0.02). The session blocking probabilities of sessions having different bandwidth requirement may be different. Therefore it is more appropriate to analyze the bandwidth blocking probability [89], which is the average of session blocking probability weighted by session bandwidth. The other performance parameter is average throughput.

Let r_i denote the current bandwidth reservation of logical link i. If the link was statically partitioned, then the admission control procedure could simply check if sum of the current reservation (r_i) and the reservation requested by new session (r), is less than or equal to the logical link capacity (C_i). The two main drawbacks of the static partitioning approach are inability to do capacity sharing and loss of statistical multiplexing.

Imagine a situation where one of the logical links is heavily loaded and its current reservation is close to its capacity, and other logical links are lightly loaded and have large free capacities. The static partitioning approach would not allow redistribution of this free capacity and would block sessions of the heavily loaded logical link even if the physical link has enough capacity to carry them. This would lower the overall utilization of the link.

Even if the session arrival rates at each of the logical links were close to each other (assuming that logical links have equal capacities), static partitioning results in higher session blocking probabilities because of loss of statistical multiplexing. The randomness in the session arrival process results in fluctuations in the reserved capacity. Even if the average reserved capacity is lower than the link capacity, there are instances when the reserved capacity approaches the link capacity and sessions arriving at these instances have to be blocked. As the link capacity increases, these fluctuations tend to have smaller impact on session blocking probabilities because of the law of large numbers. However, if the link is partitioned, the statistical variations in reserved capacity of logical links result in higher session blocking probabilities.

Given a partitioning of a link into several logical links, the Stochastic Fair Sharing (SFS) admission control decides which session reservation requests to accept and which to block. This decision process is more complex than simply checking if the current free capacity of the logical link is greater than or equal to the requested bandwidth[1]. The reservation in a logical link is allowed to exceed its capacity as long as it does not significantly affect the performance parameters of other logical links. The decision process is designed to achieve the following:

Partitioning The link capacity is partitioned into logical links.

Fair Sharing If some of the logical links are lightly loaded, their free capacity should

[1] In this chapter we discuss stochastic fair sharing in the context of link sharing, but the technique is generic enough to achieve fair sharing of a resource in a loss network [72].

be fairly redistributed to other logical links. The notion of fairness used in packet scheduling algorithms [100] can be generalized as follows. The average throughput of logical links which have blocking probability greater than a threshold (say 0.05) should be proportional to their equivalent capacity.

Statistical Multiplexing Since the traffic of all the logical links is carried on the same physical link, the bandwidth blocking probabilities while using SFS should be less as compared to static partitioning.

Isolation For sessions of a logical link it should appear as if the logical link capacity has been dedicated to them. A high session arrival rate on other logical links should not significantly increase the steady state bandwidth blocking probability (or reduce average throughput) of the logical link. The bandwidth blocking probabilities of a logical link using SFS should be comparable to that using static partitioning in the worst conditions.

If all the sessions require equal bandwidth, if the session arrival rate is a Poisson process and if a session holding time is exponentially distributed, then the session blocking probability for a link can be obtained using the well known Erlang's formula [10, 48].

$$E(\lambda, C) = \frac{\lambda^C/C!}{\sum_{i=0}^{C} \lambda^i/i!}$$

where λ is the mean session arrival rate multiplied by the mean session holding time (λ is also called arrival rate in Erlangs) and C is the link capacity (in units of number of sessions it can carry). Note that as λ and C are increased in equal proportion, the blocking probability is reduced resulting in better statistical multiplexing.

Let $\mathcal{T}_i$ and p_i be the mean throughput and blocking probability on logical link i. The mean arrival rate λ_i on the logical link is given by $\lambda_i = \frac{\mathcal{T}_i}{1-p_i}$. The equivalent capacity (C_i^e) of the logical link is defined as the capacity at which the arrival rate $\frac{\mathcal{T}_i}{1-p_i}$ results in a blocking probability of p_i.

$$C_i^e = C : E(\frac{\mathcal{T}_i}{1-p_i}, C) = p_i \tag{6.1}$$

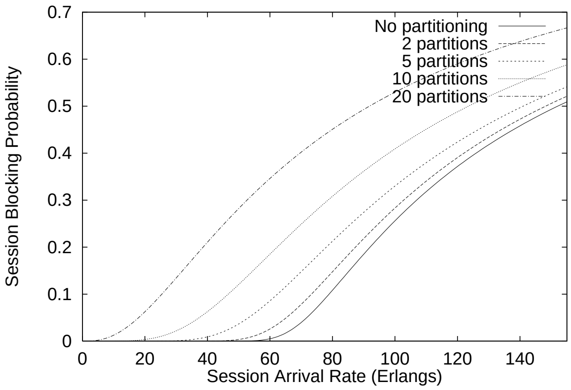

Figure 6.1: Session blocking probabilities using the Erlang's formula when link is partitioned into different number of logical links.

Note that C_i^e is unique for a given p_i and $\mathcal{T}_i$ and is always greater than $\mathcal{T}_i$. As p_i approaches 1, C_i^e approaches $\mathcal{T}_i$. According to the fairness criteria, the equivalent capacity of logical links with significant blocking probability ($p_i > p_t$, $p_t = 0.05$ say), should be proportional to their provisioned capacities (C_i). Formally,

$$\frac{C_i^e}{C_i} = \frac{C_j^e}{C_j} \forall \{\{i : p_i > p_t\} \wedge \{j : p_j > p_t\}\} \tag{6.2}$$

The term $\frac{C_i^e}{C_i}$ represents the factor by which the capacity of logical link i is increased because of unused capacity of other logical links. This quantity is always equal to 1 in case of static link partitioning, and is expected to be greater than one (representing the degree of sharing) for SFS. The variation of this quantity across the logical links (with $p_i \geq p_t$) represents a measure of unfairness of this sharing.

In a more general setting, the closed form expression for call blocking probability is very difficult to obtain and simulations are used for its estimation.

The variation in session blocking probability as session arrival rate increases is shown in Figure 6.1. We assumed a OC3 link of 155Mbps carrying video sessions of 2Mbps with a mean holding time of 3 minutes. The sessions arrive as a Poisson process with exponentially distributed session holding time. Graphs are plotted for cases when the link is partitioned into $2, 4, 10$ and 20 logical links of equal capacity. The arrival rate at each logical link is kept the same. When the arrival rate is low, the session blocking probability is almost zero and it increases as the session arrival

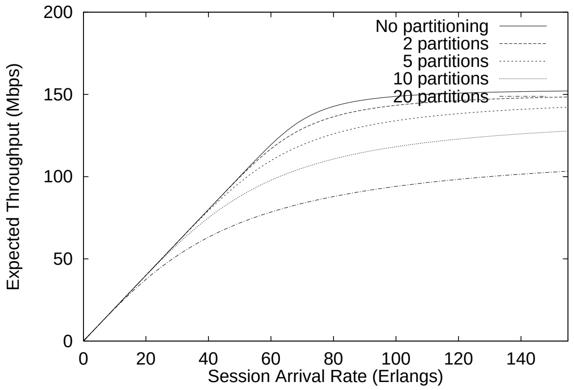

Figure 6.2: Expected throughput using the Erlang's formula when link is partitioned into different number of logical links.

rate increases. The arrival rate increases up to 155 calls per minute which is two times the rate needed to saturate the link, when the session blocking probability becomes more than 0.5. As the number of logical links are increased, the bandwidth blocking probability is increased, indicating the loss of statistical multiplexing due to partitioning. The variation in expected throughput as computed by the Erlang's formula is given in Figure 6.2. The loss of statistical multiplexing is again evident from the graphs.

SFS achieves lower bandwidth blocking probability when the arrival rate is high, at the expense of slightly higher bandwidth blocking probability when the arrival rate is low. When the arrival rate at a logical link is low, some of its free capacity is fairly redistributed to other logical links, which marginally increases its bandwidth blocking probability. However, the links with heavy load experience lower bandwidth blocking probability because of better statistical multiplexing and redistributed capacity. As a result, the overall throughput is increased significantly.

6.3 The SFS Admission Control

Let r_i be the amount of this capacity already reserved (read used) by logical link i. The normalized usage of logical link i is given by $n_i = r_i/c_i$. A logical link with normalized usage less than 1 is underutilized whereas a logical link with normalized usage greater than 1 is using more than its allocated capacity. If the given link is statically partitioned, n_i is always less than or equal to 1. In case of stochastic fair sharing, if some logical link is underutilized, other logical links may make use of its unused capacity and have a normalized usage of greater than 1.

Let the logical links be relabelled in increasing order of their normalized usage (link 1 has lowest normalized usage and link N has the highest). The link with lowest normalized usage is given the highest priority. A new session for logical link 1 is accepted if the free capacity of the physical link is greater than or equal to the bandwidth requirement of the session. A new session for logical link 2 (which has the second lowest normalized usage) is accepted only if enough free capacity (t_1) would be left in the physical link, for logical link 1 after accepting the session. This reserved capacity is called the trunk reservation for logical link 1 (a term adopted from telecommunications networks) [48]. In general, a session of logical link i is accepted if the free capacity after accepting the session is at least equal to the sum of the trunk reservations of the logical links with lower normalized usage. Formally, a new session of logical link i, with bandwidth request of r is accepted if and only if:

$$\sum_{j=1}^{N} r_j + r + \sum_{j<i} t_j \leq C \tag{6.3}$$

A session from the most heavily loaded logical link is accepted only if $\sum_{j<N} t_j$ capacity remains free after accepting this session. This scheme gives higher priority to sessions of logical links having low normalized usage and thus attempts to equalize the normalized usage of different logical links.

It has been shown in the context of telecommunications networks that even a small amount of trunk reservation gives almost absolute priority to one class of calls over the other [7, 92]. If there are only two types of calls arriving on a link and certain amount of trunk reservation (equivalent to that needed by 1-10 calls) is set aside for high priority calls, then as call arrival rate increases, the high priority calls start occupying most of the link. However, if the arrival rate of high priority calls is not large enough to saturate the link, the low priority calls occupy the rest of the link. This is shown in Figure 6.3. Sessions with bandwidth requirement of 2Mbps

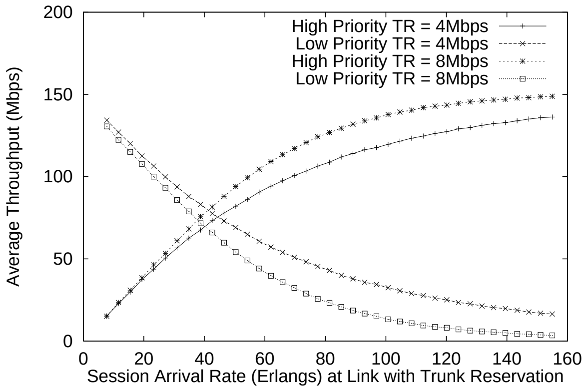

Figure 6.3: Average throughput of two links in presence of trunk reservation.

and mean holding time of three minutes arrive at a link of 155Mbps. There is a trunk reservation for high priority sessions. The arrival rate of low priority sessions is kept high at 310 Erlangs and the arrival rate of high priority sessions is varied. The graph shows that even small amount (4 Mbps) of trunk reservation gives almost absolute priority to high priority sessions.

In the proposed SFS approach, sessions in the link having the least normalized usage are given priority over the rest of the sessions. Therefore, its usage is expected to increase. As soon as its usage becomes larger than that of next link, its priority is reduced. Similarly, the link having the highest normalized usage is given the lowest priority, allowing it to only make use of the leftover capacity of all other logical links. As a result, this approach attempts to equalize the normalized usage of the logical links.

Consider a link partitioned into two logical links of equal capacity. The state of the link at any given time is represented by the current bandwidth reservation of the two logical links. Figure 6.4 shows the state space diagram of the system. The x and y axis represent the current reservation of the first and the second logical link respectively. The current state of the link can be represented by a point in the state space diagram. The shaded region represents the possible states where the system may be present at any given time. Clearly, the sum of the reservations can not be more than the link capacity as shown by the negatively sloped line on the right.

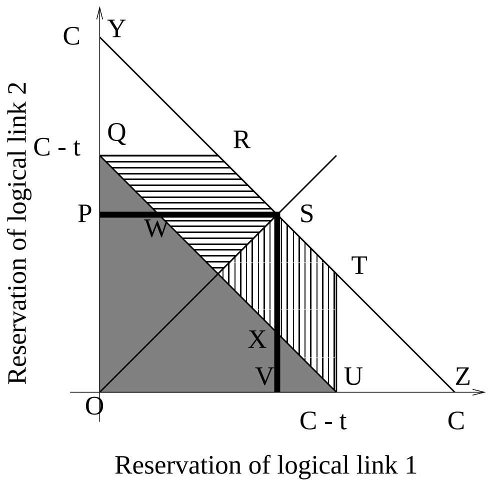

Figure 6.4: State space diagram for a link.

When a new session on first logical link is accepted, the system moves to the right in the state space diagram. Similarly, upon acceptance of a new session on second logical link, the system moves up. When sessions of first or second logical link complete, the system moves towards left or downwards. Assume that trunk reservation for both the links is same. As long the total free capacity is more than the trunk reservation, the system may move in any direction (assuming that the bandwidth requirement of sessions is small). The area shaded as gray represents this region. In the region where the available capacity is less than the trunk reservation (the area between the two slanting lines), session of the first logical link can be accepted only if the current reservation of the first logical is less than that of the second logical link. Thus, the system can move left, right or down but not upwards. This region is represented by horizontal shading. Similarly, the region shaded by vertical lines represents the state space where the system can freely move upwards but not towards the right.

If the link was statically partitioned, then its state space is limited to the rectangular region OPSV and is free to move in any direction in this region. If there is no partitioning, then the state space of the system would be the region OYZ and the system would be free to move in any direction in this region. The stochastic fair sharing approach is between these two extreme cases.

Note that the region WSX of the state space is characterized by roughly equal and moderate to high load on both the logical links. In the static partitioning approach, the system is free to move in either direction in this region, whereas in the SFS approach its movement is restricted. This may result in higher blocking probabilities for SFS if the system is expected to be in this region most of the time. Thus the performance of the SFS approach may become worse than that of static partitioning. The situation is worse if the trunk reservation is more than half the link capacity. We therefore propose a minor modification to our basic scheme described earlier.

If the normalized usage of a logical link is close to its fair share, then it is not necessary to have a large value of trunk reservation for the logical link. In this case, we reduce the trunk reservation of the logical link. Logical link i has a static trunk reservation parameter $\widehat{t_i}$ which is the maximum value of trunk reservation for the link. The fair share (f_i) of capacity for each logical link is dynamically computed. If the difference between the fair share and current reservation of a logical link is less than its static trunk reservation, then trunk reservation for the link is set to the difference. Otherwise the trunk reservation is set to the static value. Formally, $t_i = \min[\widehat{t_i}, f_i - r_i]$.

The fair share of logical link 1 is given by $f_1 = \frac{C_1}{\sum_{j=1}^{N} C_j} C$. The fair share of the other logical link is computed by redistributing the free capacity of logical links with lower normalized usage as follows:

$$f_i = \frac{C_i}{\sum_{j=i}^{N} C_j}(C - \sum_{j=0}^{i-1}(r_j + t_j))$$

The above expression is a natural extension of the fairness criteria [100] used in packet schedulers. The first term of the product in the above expression represents the fraction of residual capacity which should be given to logical link i. The second term represents the residual capacity after taking out the usage of logical links which are more lightly loaded than logical link i. Thus, if the usage of the logical link is less than its fair share, then the free capacity is fairly redistributed among heavily loaded links after keeping aside a small trunk reservation.

The state space diagram for this approach is shown in Figure 6.5. Now the system is free to move towards achieving utilization of its entire capacity as long as normalized

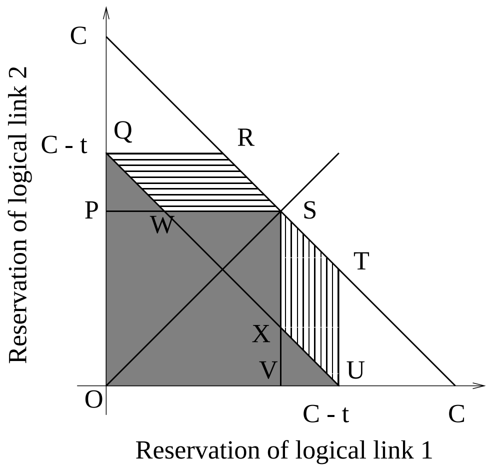

Figure 6.5: State space diagram of a link while using SFS.

usage of both the logical links is close to each other.

Note that the goals of isolation and sharing are somewhat contradictory in nature. For perfect isolation, the link has to be statically partitioned, resulting in no sharing. Redistributing even a small unused capacity of a logical link, will result in some increase in its session blocking probability. The static trunk reservation parameter (simply called trunk reservation in rest of the chapter) can be used to balance isolation and sharing. If the trunk reservation is set to zero, then the entire link capacity is shared and there is no isolation. In this case a new session is accepted if and only if the free capacity in the physical link is greater than or equal to the bandwidth requirement of the session. As the trunk reservation is increased, the sharing is reduced and isolation is improved. If the trunk reservation of a logical link is equal to its capacity then the SFS scheme emulates static partitioning (assuming that sessions have fixed bandwidth requirement, and the capacity of each logical link is an integral multiple of session bandwidth).

6.4 SFS for Multi-Hop Virtual Links

In case of virtual networks, the virtual links are realized by paths in the service provider's network. Each virtual link has a provisioned capacity allocated at the time of establishment of the link. This provisioned capacity is determined based on the long term expectation of the traffic in the virtual network and is included in the service level agreement (SLA) with the provider. In addition, each virtual link also has a current capacity which can be changed by sending *increase* or *decrease* requests in the provider network. These requests travel along the path of the virtual link. At each node in the provider network, the SFS state corresponding to each virtual link traversing the node is maintained. Variable r_i of SFS state represents the current capacity of virtual link i and C_i represents its provisioned capacity.

Upon receipt of an increase request of amount r, SFS admission control test (Eq. 6.3) is evoked to decide if the request can be accepted. If the test succeeds, SFS state at the node is updated (r_i is increased), and the request is forwarded to the next node. If the test fails, a failure indication is sent back along the same path. While this failure indication travels back, each link in the path decreases its respective r_i. If the increase request reaches the other end of the virtual link, a success message is sent back. While this message travels the reverse path, the scheduler weights are adjusted. For a decrease request, r_i's are decreased and scheduler weights are adjusted in the forward path.

Now, on a session arrival, the virtual link can simply send an increase request corresponding to the session bandwidth requirement, and accept the session if and only if the increase request is accepted in the provider network. Similarly, upon a session completion, the virtual link may send a decrease request in the provider network. While this approach would achieve weighted max-min fair sharing, it may flood the provider network with increase and decrease requests, particularly if individual VPN sessions require only a small fraction of the virtual link bandwidth. It would also result in higher call setup latencies in the VPN as the signaling message will traverse the virtual link twice.

Therefore, we set a minimum step size (C_{min}) by which the virtual link capacities can be changed. In order to prevent rapid oscillations in virtual link capacity around integer multiples of C_{min} we introduce hysteresis in sending increase and decrease requests. The decrease request is sent only if the free virtual link capacity is at least $1.5C_{min}$.

The minimum step size restriction creates the following problem. The SFS algorithm for a single link relies on the fact that session holding times are finite. If the usage of a logical link is above its fair share (because of low traffic on other logical links in the recent past), new sessions on that logical link will not be accepted, and ongoing sessions will eventually complete and release the bandwidth. However, if the minimum step size is fixed (say C_{min}) and bandwidth requirements of sessions is small as compared to C_{min}, then, instead of a decrease request, a session completion will generate some free capacity in the virtual link. At this stage, if a new session arrives it can be accepted without going through SFS admission control. Thus the virtual links may not automatically reduce their capacity even while they are operating at usage higher than their fair share.

To counter this problem, we introduce a network initiated *reduce* message. After processing an increase request at a link, the node finds out virtual links on which a small decrease request, cannot be immediately followed by a successful increase request of the same amount ($i : \sum_{j=1}^{N} r_j + \sum_{j=1}^{i-1} t_j > C$), and sends reduce request to these virtual links. After introducing hysteresis, the condition to generate a reduce request for virtual link i becomes ($\sum_{j=1}^{N} r_j + \sum_{j=1}^{i-1} t_j > C + C_{min}$). Upon receipt of the reduce request, the virtual link stops admitting new sessions until it sends a decrease request. This ensures that capacity can be quickly reclaimed from virtual links using more than their fair share.

Finally, some virtual links may still overwhelm the provider network by quickly and repeatedly sending increase requests most of which fail. We therefore impose a constraint that after an increase request fails, a virtual link is not allowed to send another increase request for a fixed interval (taken as 1 sec. for simulations).

The key ideas needed to extend SFS to multi-hop virtual links are summarized as follows.

- *Increase* and *decrease* requests in the provider network to dynamically resize virtual link capacity.
- Use of SFS admission control at each provider link for *increase* requests.
- A minimum resize capacity to limit signaling load.

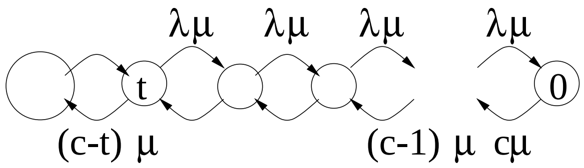

Figure 6.6: An approximate bounding model for estimating session blocking probability in worst case.

- Network initiated *reduce* request to signal highly loaded virtual links to cut down their traffic.
- Hysteresis in algorithms generating *increase*, *decrease* and *reduce* requests.

6.5 Determining Trunk Reservation Parameter

The trunk reservation parameter should balance the degree of sharing and isolation of different logical links. For the isolation property, it is sufficient to bound the session blocking probability of a lightly loaded link, when all other links have high session arrival rates. In order to compute the session blocking probability, we work with the following simplified and approximate model.

Assume that sessions have fixed bandwidth requirement, and capacities of logical links are integral multiple of session bandwidth. We scale down the capacities by session bandwidth requirement. A capacity of k means that the link can accept k sessions. Let the capacity of the physical link be C. Assume that sessions arrive independently as a Poisson process with exponentially distributed mean session holding times. Assume that all logical links except link labeled 1 have high session arrival rates. Let t be the trunk reservation for logical link 1. Since the arrival rate on all the other links is high, the sessions are expected to utilize most of the link, leaving a free capacity less than or equal to t, which is protected by the trunk reservation.

We therefore assume that the free capacity is always less than or equal to t and consider a simplified Markov chain having $t+1$ states as shown in Figure 6.6. State i represents a free capacity of i in the physical link. First, assume that the normalized usage of logical link 1 is the lowest among all the other links. Let the session arrival rate at link 1 be $\lambda_1\mu$. Let the mean session holding time be $1/\mu$. The system can move from state i to $i-1$ ($i \leq t$) only upon arrival of a session of link 1 (which has a mean $\lambda_1\mu$). However, if any of the $c-i$ sessions complete, the system can move from state i to $i+1$. If $P(i)$ represent the steady state probability that the system is in state i, then

$$P(t-i) = P(t)\frac{\lambda_1^i}{\Pi_{j=1}^{i}(C-t+j)}$$

If C is large as compared to t, then we can approximate $C-t+j$ as C. Therefore

$$P(t-i) = P(t)(\frac{\lambda_1}{C})^i$$

The session blocking probability is given by $P(0)$.

$$P(0) = \frac{(\lambda_1/C)^t}{\sum_{i=0}^{t}(\lambda_1/C)^i}$$

Now consider the case when the normalized usage of some other link becomes less than that of logical link 1. There is at least one session with a usage lower than that of link 1. Therefore, its free capacity will be redistributed and some of it will be given to link 1. As a result, the fair share of link 1 will at least be equal to its capacity. Thus, the blocking probability for a new session of this link, is expected be less than the blocking probability of the static partitioning approach ($E(\lambda_1, C_1)$). Therefore the blocking probability for link 1, for a given trunk reservation t is bounded by:

$$p(t) \leq \max\left(E(\lambda_1, C_1), \frac{(\lambda_1/C)^t}{\sum_{i=0}^{t}(\lambda_1/C)^i}\right) \tag{6.4}$$

A comparison of blocking probability as given by the above model, with the blocking probabilities obtained by simulations is given in Figure 6.7. The link of 155Mbps is partitioned into two logical links of equal capacities. We assume video sessions with a fixed bandwidth requirement of 2Mbps. The session arrival rate at one of the links is kept high at the rate of 310 Erlangs and is varied at the other link. The measured session blocking probabilities while using SFS are plotted for different values of trunk reservation. The blocking probabilities as given by the model are also plotted. The graphs indicate that the observed blocking probabilities are in good agreement with the model. When the session arrival rate is low, the Markov chain of Figure 6.6 is

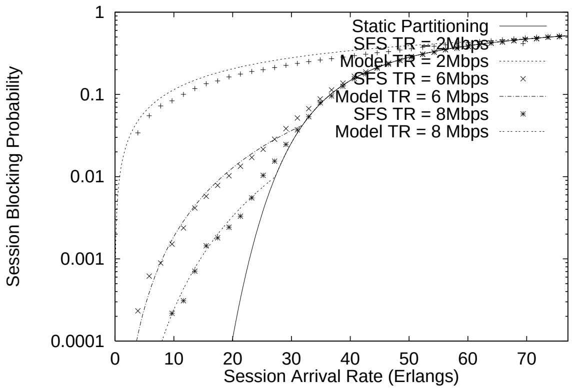

Figure 6.7: A comparison of session blocking probability as given by model with that given by simulation, in case of overload.

a good model of the system, and Eq. 6.4 gives a reasonable estimate of the blocking probability. As the arrival rate increases, the system starts behaving as if it was statically partitioned and the blocking probabilities are given by the Erlang's formula.

In order to calculate the trunk reservation for a logical link, we first select a target blocking probability p. In the extreme conditions, when there is high arrival rate on all the other logical links and the arrival rate on this link is very low, we aim for a blocking probability less than or equal to p. In case the arrival rate on this link is higher, the blocking probability will be close to that given by the static partitioning approach. Therefore we need $P(t) \leq p$ in the region where $E(\lambda_1, C_1) \leq \frac{(\lambda_1/C)^t}{\sum_{i=0}^{t}(\lambda_1/C)^i}$. Thus,

$$\begin{aligned} p &\geq \frac{(\lambda_1/C)^t}{\sum_{i=0}^{t}(\lambda_1/C)^i} \\ \Leftarrow \quad p &\geq (\frac{\lambda_1}{C})^t \\ \Leftarrow \quad t &\geq \frac{\log(p)}{\log(\lambda_1/C)} \end{aligned}$$

In order to compute a bound on t, we put λ_1 as C_1 (the arrival rate is just enough to saturate the link capacity). We get the following condition on trunk reservation:

$$t \geq \frac{\log(p)}{log(C_1/C)} \tag{6.5}$$

For a path traversing N hops, the target blocking probability may be divided by

N to get a per-hop target blocking probability. Note that the above expression is just to get a rough estimate of the trunk reservation parameter for a given worst case target blocking probability. The model is simplified, approximate and does not model the system well when the bandwidth requirements of sessions are variable. We still used Eq. 6.5, with a worst case blocking probability, p, of 0.01 to calculate the trunk reservation parameter for single link simulations. The variation in the trunk reservation parameter was small (between 3 and 5 sessions). Therefore, for network simulations, we used a fixed trunk reservation of 10 Mbps (equivalent to $t = 5$ for C_{min} = 2 Mbps). This gave us satisfactory results. If the session bandwidths are variable, we use the average bandwidth requirement of sessions to scale the capacities. Our simulations give satisfactory results for most cases. In the extreme case when the physical link of 155Mbps was partitioned into 20 logical links, the calculated trunk reservation was 3Mbps which was less than the maximum bandwidth requirement of a session (3.75Mbps). This resulted in poor blocking probabilities of lightly loaded links in case of overload.

6.6 Single Link Simulations

SFS critically relies on the assumptions that (a) bandwidth requirement of individual sessoins is small as compared to the link capacity, (b) most sessions have small holding time, and (c) the session arrival process is not very bursty.

Accurate models for the arrival process of sessions with QoS requirements are not available. The only models available are for call arrivals in telecommunications networks where researchers have extensively used Poisson call arrival model (with arrival rate as a function of the time of day) with exponential holding time (with a mean of about 3 minutes). The assumptions required for SFS are definitely true in telecommunications networks.

These assumptions also seem reasonable for future Integrated Services Networks. For instance, the bandwidth requirement of MPEG video stream is between $1 - 6$ Mbps, which is reasonably small as compared to link speeds of 155 Mbps to 2.4 Gbps. The average throughput of a typical web connection is also small (4 Kbps

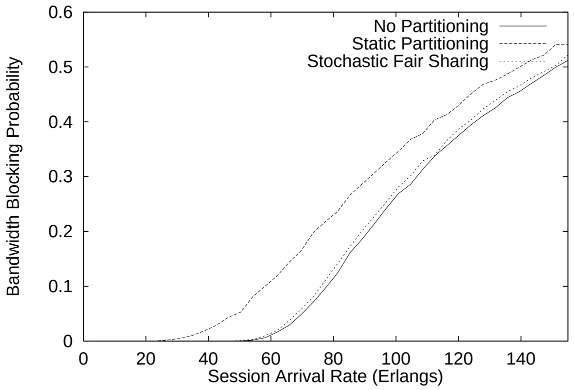

Figure 6.8: Bandwidth blocking probabilities under same arrival rate on all logical links.

to 128 Kbps) [4] as compared to the link speeds. Though the holding time for web connections is shown to be heavy tailed [23], most of the connections are still short-lived. The session arrival process is also not as bursty as the data arrival process.

We therefore simulated using the most simplistic model. We assume that sessions arrive as a Poisson process having exponentially distributed holding times with a mean of three minutes. The bandwidth requirement of these sessions vary uniformly at random from 250Kbps to 3.750Mbps in increments of 250Kbps. The mean session bandwidth is 2Mbps. Though these assumptions do not accurately model the session arrival process, we do expect them to give a glimpse into the SFS performance. We assume an OC3 link of 155 Mbps, partitioned into logical links.

6.6.1 Statistical Multiplexing

Figures 6.8 and 6.9 show how SFS can achieve more statistical multiplexing. The physical link is partitioned into five logical links. The session arrival rate in each of the links is kept the same and the total session arrival rate is increased till 155 Erlangs (two times the rate required to saturate the link). The bandwidth blocking probability, which is the same for each logical link, is measured and plotted for different schemes. “No partitioning” refers to the scheme in which a new session is accepted if the physical link has enough free capacity. In “Static partitioning”, a new session is

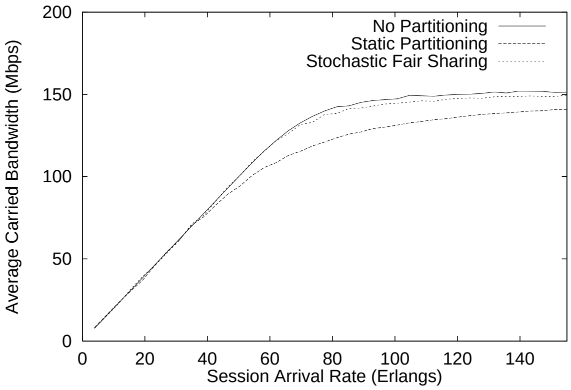

Figure 6.9: Average throughput under same arrival rate on all logical links.

accepted only if free capacity of its logical link is greater than or equal to its bandwidth requirement. The bandwidth blocking probability and the average throughput of SFS is close to that of the "no partitioning" approach which is the best achievable. Therefore, SFS regains most of the statistical multiplexing lost by fixed partitioning.

6.6.2 Isolation

The performance parameters of a logical link should not degrade in the presence of overload on the other logical links. To see this, we have simulated two overload scenarios. We partition the physical link into five logical links and assign high session arrival rates on four of the links. The session arrival rate on other logical link (which is lightly loaded) varies from 0 to 31 Erlangs (two times the rate needed to saturate the logical link). In the "constant overload" scenario, the session arrival rate on overloaded logical links was kept to 310 Erlangs, whereas in the "proportional overload" scenario the arrival rate on the overloaded links was kept ten times the arrival rate on the lightly loaded link. If there is adequate isolation, the bandwidth blocking probabilities of the lightly loaded link would not be much worse than that of the static partitioning approach.

Figure 6.10 plots the bandwidth blocking probability of the lightly loaded link as its session arrival rate increases. Note that the blocking probability in the presence of overload is very close to that of static partitioning. This shows that the SFS approach

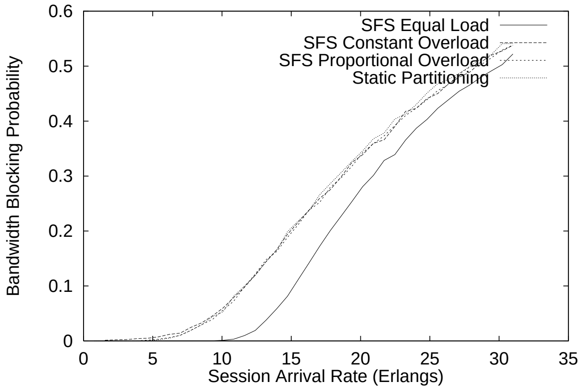

Figure 6.10: Bandwidth blocking probability of lightly loaded link, when other links are overloaded.

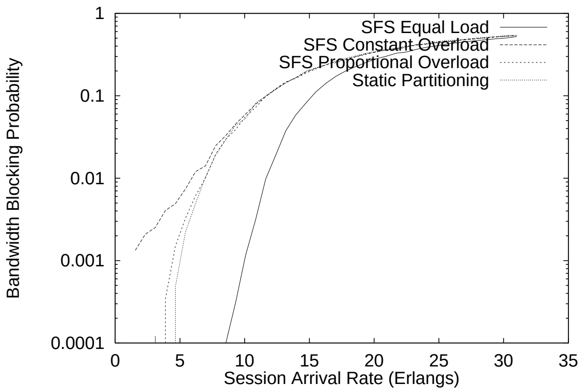

Figure 6.11: Bandwidth blocking probability of lightly loaded link, when other links are overloaded.

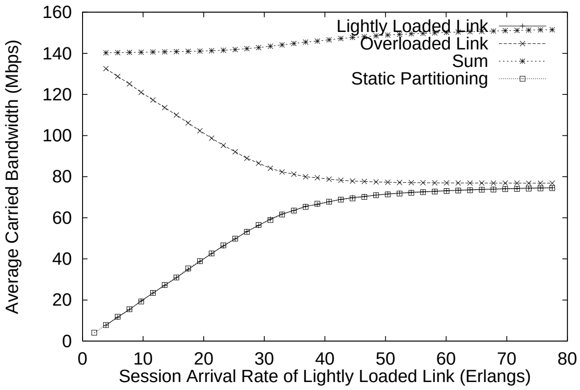

Figure 6.12: Average throughput of the two logical links.

achieves isolation even in the presence of overload in other links. The same graph is shown in log scale in Figure 6.11. Note that when the arrival rates are very low, and the blocking probability is 0.01 (the target blocking probability) or less, then the blocking probability of the SFS approach does not reduce as fast as in static partitioning. In this range of operation, the free capacity of lightly loaded links is redistributed to overloaded links resulting in higher blocking probability. However, if the arrival rates on all the links is equal, then blocking probability is reduced.

6.6.3 Fair Sharing

In order to illustrate sharing, we assume that the physical link is divided into two logical links of equal capacity. Again, one of the links is lightly loaded whereas the other link is overloaded. The overloaded link has a constant session arrival rate of 310 Erlangs, which is eight times the rate required to saturate the physical link. The arrival rate in the other logical link varies from 0 to 77.5 Erlangs. Figure 6.12 shows the variation in the average throughput as the arrival rate in the lightly loaded link increases. At zero session arrival rate, most of the free capacity is used by the overloaded link. However as the session arrival rate increases, the average throughput of the lightly loaded link increases. The graph showing average throughput of the lightly loaded link overlaps with the graph showing the same if static partitioning was used, indicating that there is virtually no loss of throughput at the lightly loaded

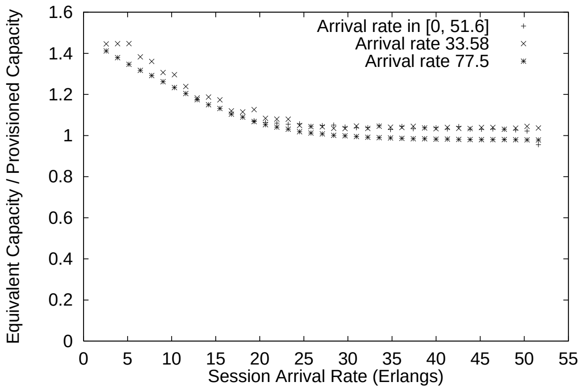

Figure 6.13: Graphs showing fair sharing of residual capacity while using SFS.

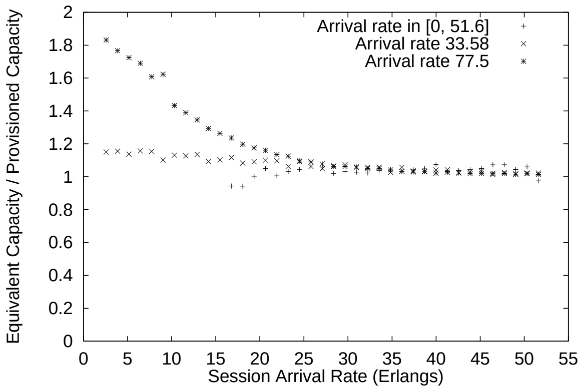

Figure 6.14: Sharing of residual capacity while using virtual partitioning.

link even if its capacity is redistributed to other links. The total average throughput while using static partitioning varies from 77 Mbps to 155 Mbps. The total average throughput while using SFS increases from 140Mbps to 155Mbps, indicating that the trunk reservation of the lightly loaded link is also utilized as its arrival rate increases.

To demonstrate fairness of sharing, we assume that the physical link is partitioned into three logical links of equal capacity. One of the logical links is overloaded with an arrival rate of 77.5 sessions per minute (three times the nominal load), the other link is loaded with an arrival rate of 33.58 Erlangs (1.3 times the nominal load) and the arrival rate at the third link is varied from 0 to 51.6 Erlangs (two times the nominal load). The ratio of equivalent capacity to provisioned capacity for links having blocking probability greater than 0.05 is plotted in Figure 6.13. The same

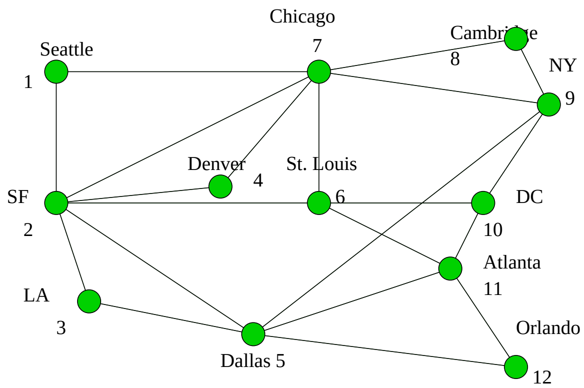

Figure 6.15: Topology of simulated network.

graph is plotted for the virtual partitioning scheme [12] in Figure 6.14. Points with zero ratio indicate that the bandwidth blocking probability is smaller than 0.05. In the case of virtual partitioning, the free capacity is taken almost entirely by the logical link having high arrival rate. When the arrival rate on one of the logical links is low, there is a large free capacity. In this range, the equivalent capacity of the link having an arrival rate of 77.5 Erlangs, is 1.8 times its provisioned capacity whereas the equivalent capacity of the other link with arrival rate 33.58 Erlangs is only 1.15 times its provisioned capacity. As the arrival rate on one of the links increases, the free capacity decreases and eventually becomes equal to zero. In this region of the graph, equivalent capacity of virtual links approach their provisioned capacity. The graphs for SFS show that the ratio of equivalent capacity to provisioned capacity is very close to each other for the entire range of arrival rate on one of the logical links. This indicates that SFS redistributes the residual capacity fairly among the logical links in accordance with our fairness criteria.

6.7 Simulations for a Network

We report the results for simulations carried out on a twelve node approximate AT&T Worldnet topology (also used in [27]) as shown in Figure 6.15. Each link is assumed to be of capacity 2.4 Gbps and propagation delay 20ms. We setup a leased virtual link between every pair of these nodes. Therefore, there are 132 unidirectional virtual links. A fixed trunk reservation of 10 Mbps and a minimum increase/decrease step size of 2 Mbps was used.

6.7.1 Max-min Fairness

We assume that all the 132 virtual links have the same provisioned capacity. We assume, as in Section 6.6, that sessions with rate uniformly distributed between 0.25 Mbps and 3.75 Mbps in increments of 0.25 Mbps, randomly arrive (as a Poisson process) at these virtual links. The session holding time is exponentially distributed with a mean of three minutes ($1/\mu = 3$min.). A load of 1 on a virtual link implies a session arrival rate of $f\mu/r$, where f is its max-min fair capacity, r is its mean session bit-rate. In the (0.0-2.0) scenario, the load on each virtual link is set uniformly at random between 0 and 2. In the (0.5-1.5) scenario, the load on virtual links is set uniformly at random in the range 0.5 and 1.5. In the 5% misbehaving scenario, 5 percent of the virtual links misbehave with a load of 10. The other virtual links have a load of 1.

We compared SFS with the simplest first come first serve (FCFS) scheme. In FCFS, each virtual link sends increase and decrease requests as in SFS, but the admission control procedure in the provider network grants these requests in first come first serve order (i.e., as long as there is free capacity on the link).

Simulations were run for 2000 simulated seconds. In the end, each virtual link was classified as bottlenecked or non-bottlenecked, depending upon whether or not its session blocking probability was more than a given threshold ($p_{th} = 0.05$). The average throughput of non-bottlenecked virtual links was subtracted from the link capacities in its path. For rest of the virtual links, weighted max-min fair share was computed using the residual capacities. Now, using the Erlang's formula, the throughput of non-bottlenecked virtual links was adjusted upwards, depending upon their session blocking probabilities, to reflect their equivalent capacity. The fairness ratio represents the ratio of equivalent capacity to max-min fair capacity. The experiment was repeated a number of times with different random seeds.

Figure 6.16 shows the cumulative distribution function of the fairness ratio of all

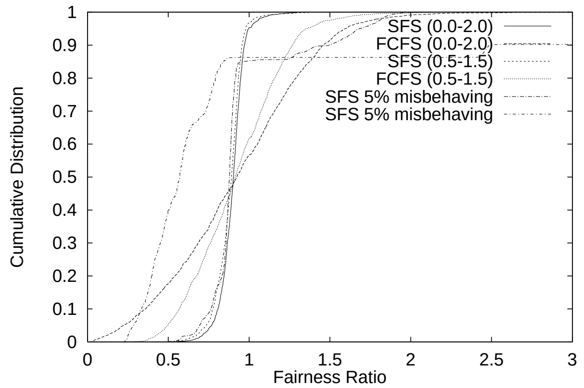

Figure 6.16: Cumulative distribution of fairness ratio for multi-hop virtual links, with and without using SFS.

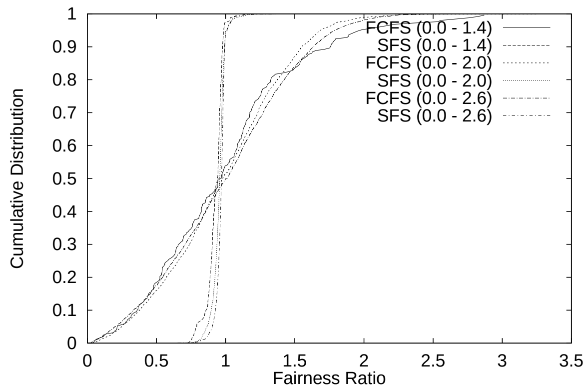

Figure 6.17: Cumulative distribution of fairness ratio for multi-hop virtual links, with and without using SFS (constant bandwidth sessions).

the virtual links for different loading scenarios. From the figure, it is evident that SFS converges reasonably well to the weighted max-min fair share in the provider network. For 90% of the cases, the fairness ratio of SFS is between 0.75 and 1.00, for (0.0-2.0) and (0.5-1.5) scenarios. Note that the Erlang's formula is only applicable for constant bandwidth sessions. Therefore computing the equivalent capacity using the Erlang's formula even in the presence of variable bandwidth sessions, introduces a small error. A significant portion of the spread in fairness ratio can be attributed to this error. When constant bandwidth sessions are used in the simulations the fairness ratio remains between 0.90 and 1.05 for about 95% of the cases as shown in Figure 6.17.

As the variation in load increases from (0.5-1.5) to (0.0-2.0), the spread in fairness ratio for FCFS increases, whereas for SFS this spread remains nearly the same. The fairness ratio ranges from 0.09 to 3.02 for FCFS and from 0.43 to 1.38 for SFS.

In the case of 5% misbehaving virtual links, the FCFS allocates large capacities to the misbehaving virtual links, resulting in a significantly lower fairness ratio of most of the well behaved virtual links. In case of SFS, the fairness ratio of well behaving virtual links remains nearly same as before, indicating that they are well protected from the misbehaving virtual links. The range of fairness ratios for FCFS and SFS for this scenario are 0.22 to 5.23 and 0.40 to 2.08 respectively.

Another measure of fairness has been proposed in [65]. The fairness index is defined as $I = (\sum_{i=1}^{n} f_i)^2/(n(\sum_{i=1}^{n} {f_i}^2))$, where f_i is the fairness ratio of virtual link i and n is the number of virtual links. A fairness index of 1 implies perfect fairness and fairness index 0 implies gross unfairness. The fairness index ranges of SFS for the three scenarios are $(0.989 - 0.997), (0.993 - 0.998), (0.963 - 0.974)$. These ranges for the FCFS are $(0.938 - 0.985), (0.869 - 0.973), (0.644 - 0.658)$. According to this index of fairness, SFS achieves close to perfect fairness for all the scenarios, whereas FCFS performs significantly worse in the 5% misbehaving scenario.

All these results indicate that SFS achieves its primary objective of protection and max-min fair sharing in a generic network.

Signaling load	Percentage of Low Bandwidth Traffic			
(messages / sec)	0	30	60	90
Avg. (out) v-links	1.36	1.33	1.33	1.60
Max. (out) v-link	5.66	6.00	6.50	8.42
Avg. (VPN) v-link	2.13	21.59	41.03	59.9
Max. (VPN) v-link	9.41	94.23	178.62	263.18
Avg. over routers	76.20	74.24	74.09	88.48
Max. over routers	120.37	118.06	118.57	144.71
Addnl. setup latency (ms)				
Avg. over v-links	29.27	2.78	1.32	0.92
Max over v-links	76.77	7.63	4.13	2.96

Table 6.1: Signaling load and additional call setup latency for different mix of high-bandwidth and low-bandwidth traffic.

6.7.2 Signaling Load

Some fraction of low bandwidth traffic (64Kbps sessions) was mixed with the high bandwidth traffic, and the signaling load on virtual links and routers was measured. Table 6.1 shows the results for one simulation run of 2000 seconds. On each virtual link, the number of *increase, decrease* requests sent and *reduce* requests received was measured. The table indicates that the average rate of these signaling requests remains fairly constant in the range 1.33 to 1.60, even if the fraction of low bandwidth sessions in the VPN increases. The number of average VPN signaling requests processed by virtual links increases to up to 59.9 as the percentage of low bandwidth sessions increase. However, most of these requests are filtered by the virtual links and do not result in a signaling message in the provider network. The rate of signaling message in the provider network remains fairly independent of the traffic mix in the VPN. The maximum signaling requests were usually generated by the Denver-Chicago (4-7) virtual link (between 5.66 and 8.14 messages per second for different traffic mixes). The table indicates that the average signaling load on provider routers using SFS is also fairly constant over different traffic mixes (between 74.09

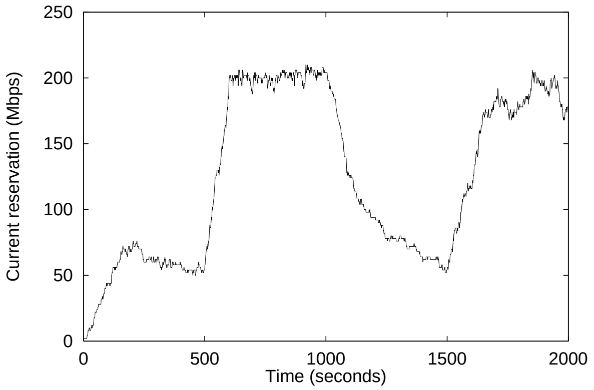

Figure 6.18: Variation of current reservation on a virtual link, when arrival rate was changes at 500, 1000 and 1500 seconds.

and 88.48), and is quite manageable[2]. The maximum signaling load was on Chicago and SF routers (up to 144.71 messages per second) which also have maximum number of links connected. This indicates that using SFS in a large network is quite feasible and does not require a large signaling overhead for the increase, decrease and reduce requests.

6.7.3 Other Parameters

Use of SFS may introduce additional session setup latency in VPN because the setup request may trigger an increase request on a virtual link and then the setup request has to wait till the response of this increase request comes back. Table 6.1 also shows the average and maximum of additional session setup latencies. If the traffic mix consists of high bandwidth traffic only, then approximately two session arrivals on a virtual link result in one increase request. So, the additional setup latency is about half the round-trip time on virtual links. As the fraction of low bandwidth session increases, this setup latency goes down significantly. Only a very small fraction of low bandwidth sessions generate an increase request and therefore have low average latency.

[2]Note that SFS admission control is evoked on less than 1/4th of these messages (only on increase-fwd out of increase-fwd, increase-bwd, decrease-fwd, decrease-bwd and reduce).

Adaptability is another important parameter while using SFS in a network. What happens if session arrival rate on one of the virtual links changes? How quickly can SFS adapt to change in traffic patterns? Figure 6.18 gives the answers. It shows the variation of reserved bandwidth on Seattle-Orlando virtual link with time. We simulate with only high-bandwidth sessions in the network, and randomly change the session arrival rate on the Seattle-Orlando virtual link at time $T = 500, 1000$ and 1500 sec. The initial arrival rate on the virtual link was less than that needed to saturate its max-min fair capacity. Therefore, the reservation level steadily increased to 70 Mbps and then varied randomly till $T = 500$. The blocking probability was 0 in this interval. At $T = 500$ the arrival rate became large enough to saturate the max-min fair capacity. So the average reservation increases rapidly and stabilizes around a max-min capacity of 200Mbps. The blocking probability is 0.3 in this interval. At $T = 1000$, the arrival rate becomes small again. Now the average reservation reduces exponentially to 56Mbps in till $T = 1500$. The blocking probability in this region is 0.025. At $T = 1500$, the arrival rate is increased again, but is insufficient to saturate the max-min capacity. Therefore the reservation varies between 170Mbps and 200Mbps. The blocking probability in this region is again 0. If the arrival rate is increased, it takes about 180 seconds (the mean call holding time) for the reservation level to reach close to its steady-state value. If the arrival rate is reduced, the decay in reservation is slow and exponential (which can be attributed entirely to the exponential holding time of sessions). Thus, the rate of adaptation of SFS is primarily limited by the mean session holding time.

The trunk reservation used by SFS may lead to some bandwidth wastage. To estimate this, we measured the total number of bits carried while using SFS and compared it with FCFS (no partitioning, increase request served in FCFS order). The average throughput per virtual link of SFS for scenarios (0.5-1.5), (0.0-2.0) and "misbehaving" were 404 Mbps, 373 Mbps and 432 Mbps respectively. The average throughputs of FCFS for these scenarios were 409 Mbps, 379 Mbps and 429 Mbps respectively. For the "misbehaving" scenarios, SFS outperforms FCFS whereas in the other two scenarios the average throughput of SFS is within 98% of FCFS. Thus, the bandwidth wastage in SFS is small as compared to the potential resizing gains (around 2 as reported in [27]), while the protection and fairness benefits are large which is essential for any resizing approach to work in practice.

6.8 Future Work

A different technique to do fair sharing based on the idea of dynamic threshold [18] can be designed. For every logical link i, a capacity threshold is computed from the unused link capacity as follows:

$$T_i = \frac{C_i}{C} f\left(C - \sum_{j=1}^{N} r_j\right) \tag{6.6}$$

where $f(.)$ is a monotonic positive function. A new session of bandwidth r on logical link i is accepted if and only if:

$$C_i + r \leq T_i \tag{6.7}$$

The capacity threshold (T_i) for each logical link is proportional to the provisioned capacity (C_i) of the logical link. Thus the capacity distribution is fair in the steady state. In the simplest scheme, the capacity threshold may be set to a multiple of the unused capacity as follows:

$$T_i = \alpha \frac{C_I}{C}\left(C - \sum_{j=1}^{N} r_j\right) \tag{6.8}$$

The capacity left unused by this scheme in the best case (when $r_i = T_i$) will be $\frac{C}{1+\alpha}$.

The admission control decision with this scheme can be made very quickly as it involves simple multiplication, division and comparison.

In case of a network, the admission control may be performed using the dynamic threshold as given by Eq. 6.6 and 6.7. If the current reservation of a virtual link becomes greater than its threshold (T_k) at any node in its path, the reduce message containing the threshold may be sent. The virtual link end point, on receipt of a reduce message resizes its capacity to the threshold (T_k) contained in the message.

The design and performance analysis of this scheme and other similar schemes and their comparison with SFS is a topic of future research.

An important practical question left unanswered in the discussion is the following. Why would a VPN customer ever reduce the capacity of his/her virtual links (either voluntarily or upon the receipt of a reduce message)? In order to reclaim the unused capacity of a virtual link, either some (financial) incentive should be provided to the customer, or the behavior of a *compliant* customer should be defined and included as a part of the service level agreement (SLA) between the customer and the service provider. However, a reasonable definition of compliant customer itself seems difficult to obtain. Moreover, detecting non compliant customer is also a difficult task.

The alternative of providing financial incentive appears more promising. With the advent of electronic commerce (e-Commerce), the enormous book-keeping and billing process required could be automated. The network management protocol may be augmented to carry out limited financial transactions also. For instance, the response to a decrease message could carry a discount in the form of electronic cash (or a future bandwidth option) and an increase message should carry the required cash (or bandwidth option) to buy the required capacity. Another option could be to carry out an online real-time auction of the available network capacity. The viability, format and structure of such incentives is a matter of future investigations.

6.9 Conclusions

In this chapter we presented a new scheme called stochastic fair sharing (SFS) to carry out fair link sharing and max-min fair sharing among leased virtual links of VPNs. We argue that fair scheduling algorithms like (WFQ) carry out excess capacity redistribution over time-scales of packet transmission time and therefore fair sharing gains can only be utilized by "elastic" non real-time traffic. In the previous chapter we showed that in the case of virtual links of virtual networks, due to the output burstiness constraint, even the non real-time traffic may not be able to make full use of all the free capacity available on physical links. Even if separate tunnels for real-time and best-effort traffic are created, the sharing is limited only in the "elastic" portion of the best-effort traffic.

The proposed SFS scheme carries out fair sharing by dynamically resizing virtual link capacity at the granularity of session arrivals. The redistributed capacity is available for session lifetime, and thus real-time sessions can make use of it by appropriately adjusting the weights in the underlying schedulers.

As a result of sharing, the unused capacity of lightly loaded virtual links is redistributed to heavily loaded virtual links, resulting in higher aggregate throughput (and possibly higher revenues). This capacity redistribution is carried out in such a way that blocking probability of lightly loaded logical links is not significantly affected. Therefore, the lightly loaded logical links can quickly regain their share of link capacity if their session arrival rate is increased. This ensures adequate protection to lightly loaded virtual links, which enables them to release their unused capacity for reallocation by the network.

The key components of the SFS scheme are SFS admission control, which decides whether a capacity resizing request should be accepted, the algorithm to generate an explicit reduce request to request down-sizing of some virtual links, and the network level signaling protocol which carries out capacity resizing with minimal overheads.

In link sharing environment, only the SFS admission control is required. This makes it very simple to implement. In case the arrival rates at logical links are proportional to their link capacities, no sharing is possible. In this case SFS results in a lower blocking probability because of better statistical multiplexing of aggregated traffic. We illustrate these facts by carrying out simulations under simplistic assumptions of Poisson session arrivals and exponentially distributed session holding time. In case the load on logical links is uneven, SFS carries out fair redistribution of unused capacity. We give simulation results to supplement this. We also compare SFS with virtual partitioning scheme [12] which does not carry out this redistribution fairly.

In a network environment, other parameters like signalling load, increase in setup latency also become important. We describe some simple algorithms to generate the capacity resizing requests, network initiated reduce requests which may be used with SFS admission control to ensure that the signaling load and the setup latencies are also low.

We carry out simulations for the network environment for different traffic mixes. The simulations indicate that SFS achieves max-min fair sharing. With constant bandwidth sessions of 2 Mbps, 94 percent of virtual links achieve an equivalent capacity within -10 to $+5$ percent of their max-min fair capacities. With variable bandwidth sessions (0.25 Mbps to 3.75 Mbps), 90 percent of virtual links converge within 25 percent of their max-min fair share. We attribute a part of this spread to inaccuracy in computing the equivalent capacity. The fairness index [65] is shown to vary between 0.96 and 0.99 even in the extreme situations.

The signaling load to implement SFS is shown to be small (between 75 and 90 messages per second per router) and relatively insensitive to the traffic mix. The load remains in this range even if a high fraction of low bandwidth sessions is present. The bandwidth penalty for using SFS was found to be less than 2%. The rate of adaptation of SFS is of the order of mean session holding time. If the arrival rates are changed suddenly, our simulations indicate that SFS converges to the changed scenario within two mean session holding times.

SFS is ideal for achieving fair link sharing in telecommunication networks and ATM networks. It may be used to carry out dynamic bandwidth allocation of virtual paths in ATM networks. In addition to achieving fair link sharing, SFS may be used in virtual networks to carry out dynamic bandwidth allocation of virtual links. Thus, SFS can provide fair resource sharing in virtual networks. In the context of differentiated service in IP based networks, SFS may be integrated with bandwidth brokers to implement fair resource sharing in networks. The SFS technique is generic enough to carry fair sharing of resource in a generic loss network. A simple extension of SFS can carry out the resource partitioning hierarchically.

Chapter 7

Concluding Remarks

In the first part of this thesis we examined the three important aspects of traffic management in Integrated Services Networks: traffic characterization, admission control and scheduling. In the second part we examined these aspects in the context of Integrated Services Virtual Networks.

Since video traffic is expected to have a significant share of the real-time traffic we focused on video traffic characterization. Most of the work on video traffic characterization either used small traces, or analyses traffic generated by only one compression algorithm. We have attempted to be more exhaustive in this respect. We have looked at traffic generated by three coding algorithms namely JPEG, MPEG and compression algorithm of the software NV. We have considered one to two hour long traces of five video sequences of different types, ranging from a lecture in a classroom to a basketball match.

We characterized burstiness of video traffic at different time scales using the burstiness function. We found that the constant quality video is inherently bursty in the long term because of changes in its scene complexity and the amount of information present in different scenes. The MPEG compression algorithm adds short term burstiness which is specific to the algorithm. Similarly the compression algorithm of the software NV adds the algorithm specific short term burstiness, but removes the long term burstiness by reducing the scene quality of complex scenes.

Traditional work on video traffic characterization is a two step process. First, a statistical model of the video traffic is built and then admission control tests for this model are derived. We adopted a more direct approach where the traffic is characterized using the practical models already in use for admission control tests.

We compared the X_{min}, X_{avg}, I traffic model with the leaky bucket traffic model and found that the leaky bucket model results in better network utilization. We also discussed some insights that help us choose a leaky bucket traffic descriptor for video traffic. The knee point in the leaky bucket traffic descriptor graph of JPEG and MPEG video traffic turns out to be a good candidate for its traffic descriptor.

The earlier admission control tests for deterministic guarantees were suboptimal. We suggested new optimal admission control tests for the EDF scheduling algorithm. Using these tests and video traffic traces we analyzed the performance of deterministic QoS guarantees. We showed that video traffic alone could result in high utilization of the network.

Together with traffic modeling and connection admission control algorithms, schedulers are the most important components for providing QoS guarantees. While connection admission control algorithms reserve resources during connection establishment time, packet scheduling algorithms allocate resources according to the reservation during data transfer.

Many of the scheduling algorithm proposed in the literature, were first designed for scheduling packets of variable sizes and were complex [26, 131, 128]. Some of them were later adopted for scheduling ATM cells [122]. However the inherent complexity remains and typical operations needed to schedule a cell are addition, multiplication and division. In addition they also need a multi-level priority queue. Among the schedulers optimized for scheduling ATM cells, some have poor delay and fairness properties [87], some are not scalable for fine rate granularity [99] and some may need to over-allocate rate resulting in poor utilization of link bandwidth [68].

We developed a new scheduling algorithm called the recursive round robin scheduler (RRR) for very high speed networks. We outlined a specialized high speed hardware implementation of the RRR scheduling algorithm assuming that the scheduler works on fixed size packets. We also presented a method to speedup general software implementation so that it takes only four to six memory accesses to schedule a packet. We analytically analyzed the delay and fairness properties of the scheduler. We showed that the scheduler has all the properties needed to provide QoS in an integrated services network. We then adapted the scheduler for variable sized packets and briefly discussed the properties of the variable size packet scheduler.

In the second half of this thesis we focused on Integrated Services in virtual networks. We showed why traditional scheduling algorithms are not suitable for providing QoS guarantees in virtual networks. We introduced a term called output burstiness and showed that schedulers having low output burstiness in virtual networks have good delay properties. We extended the theory of latency rate servers for virtual networks and using this theory presented two variants of the RRR scheduling algorithm for virtual networks.

The output burstiness constraint limits the rate at which traffic may be sent on virtual links of a virtual network. As a result, some packets may have to wait in queues even while the physical link is idle. This reduces the overall throughput of the network. One of our suggested solutions was to use separate virtual links for real-time and best-effort traffic. Bounded output burstiness is a constraint only for real-time virtual links which require strict delay guarantees. Since there is a closed-loop congestion control algorithm for best-effort traffic, best-effort virtual links may send traffic at high rate utilizing all the available capacity of the physical links. This introduces difficulty in handling best-effort traffic with minimum bandwidth guarantees. To support this, the virtual link capacity needs to be partitioned for real-time and best-effort links. This limits the sharing benefits only to the "elastic" portion of the best-effort traffic.

We finally proposed a capacity resizing approach to enhance sharing among real-time traffic. In this solution, the capacities of virtual links are dynamically adjusted depending on their traffic demand and load on the physical network. This kind of approach has also been proposed recently by other researchers [27, 53] and shown to result in significant gains (factor of 2 to 3). Protection and fairness are two key properties needed for such a resizing approach to work in practice. Our proposed technique called stochastic fair sharing (SFS) provides both fairness and protection.

SFS has two major components. The SFS admission control procedure decides which resize requests to accept such that the link capacity is shared in a fair manner. The second component is algorithms to generate the resize requests and protocol to carry them across the network. Some of the resize requests (*increase, decrease*) are generated by the virtual links and some (*reduce*) by the network.

We showed how SFS can be used in a link sharing environment, where a single link is partitioned among several classes. In this case only the SFS admission control algorithm is used. For sessions of every traffic class, SFS decides whether to accept the session or not depending on the current utilization of the class and the overall utilization of the physical link. After a session is accepted, the underlying scheduler weights are adjusted. SFS ensures that if a class is lightly loaded, its residual capacity is redistributed to heavily loaded classes in a fair manner. SFS also ensures that the blocking probability of lightly loaded classes is kept low so that its perceived QoS (in terms of blocking probabilities) is not significantly affected and the class can quickly regain its fair share as its session arrival rate increases.

We showed how SFS can be used in the case of multi-hop virtual links of VPNs. In this case the virtual links send capacity resizing requests which are admission controlled in physical network using SFS. We presented simulation results for a twelve node network indicating that using SFS, equivalent capacity of virtual links converge to their max-min fair capacity, with a fairness index of 0.97 even in extreme situations. When only constant bandwidth sessions of 2 Mbps were present, the equivalent capacity of 94% of virtual links converged within -10% to $+5\%$ of their max-min fair share. When the session rates were uniformly distributed between 0.25 Mbps and 3.75 Mbps, equivalent capacity of 90% of virtual links was within 25% of their max-min fair share. A part of this spread was because of inability to compute the exact equivalent capacity of virtual links.

The average signaling load of the protocol was less than 100 messages per second per router and was found to be insensitive to the traffic mix. Thus, each session arriving on a virtual link does not generate a capacity resize request on the virtual link. The bandwidth penalty for using SFS was found to be less than 2%.

SFS is ideal for achieving fair sharing in telecommunications networks and ATM networks. It may also be used in integrated services and differentiated services based

IP Virtual Private Networks (VPNs). The SFS technique is generic enough to carry fair sharing of resource in a generic loss network. A simple extension of SFS can carry out the resource partitioning hierarchically.

Thus, SFS alongwith schedulers with bounded burstiness may be used to provide Integrated Services in virtual networks.

Bibliography

[1] User network interface specifications 3.1. ATM Forum, Prentice Hall, 1993.

[2] Private network to network interface specification 1.0. ATM Forum, Prentice Hall, March 1996.

[3] Hari Balakrishnan, Srini Seshan, Venkata Padmanabhan, Mark Stemm, and Randy H. Katz. TCP behavior of a busy internet server: Analysis and improvements. In *Proceedings of INFOCOM*, San Francisco, CA, USA, March 1988.

[4] Hari Balakrishnan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. Analyzing stability in wide-area network performance. In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 1997.

[5] Anindo Banerjea and Srinivasan Keshav. Queueing delays in rate controlled networks. In *Proceedings of INFOCOM*, San Francisco, April 1993.

[6] Anindo Banerjea and Bruce Mah. The real-time channel administration protocol. In R.G. Herrtwich, editor, *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 160–170, Heidelberg, Germany, November 1991. Springer-Verlag.

[7] N. G. Bean, Richard J. Gibbens, and S. Zachary. Asymptotic analysis of single resource loss systems in heavy traffic, with applications to integrated networks. *Advances in Applied Probability*, 27:273–292, March 1995.

[8] Jon C. R. Bennett and Hui Zhang. WF2Q: worst-case fair weighted fair queueing. In *Proceedings of INFOCOM*, pages 120–128, San Fransisco, California, March 1996.

[9] Jon C. R. Bennett and Hui Zhang. Why WFQ is not good enough for integrated services networks. In *Proc. of The 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 96)*, Shonan Village International Conference Center, Zushi, Japan, April 1996.

[10] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks.* Prentice Hall, Englewood Cliffs, NJ, USA, 1992.

[11] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated services, October 1998. Internet Draft <draft-ietf-diffserv-arch-02.txt>.

[12] Sem C. Borst and Debasis Mitra. Virtual partitioning for robust resource sharing: Computational techniques for heterogeneous traffic. *IEEE Journal on Selected Areas of Communications*, 16(5):668–678, June 1998.

[13] M. Butto, E. Cavallero, and A. Tonietti. Effectiveness of the leaky bucket policing mechanism in ATM networks. *IEEE Journal on Selected Areas of Communications*, 9(3):335–342, 1991.

[14] M. Casoni and J. S. Turner. On the performance of early packet discard. *IEEE Journal on Selected Areas of Communications*, 15(5):892–902, June 1997.

[15] Traffic control and resource management in B-ISDN. CCITT Draft Recommendation I.371, December 1991.

[16] Mun Choon Chan, H. Hadama, and R. Stadler. An architecture for broadband virtual networks under customer control. In *Proceedings of NOMS '96 - IEEE Network Operations and Management Symposium*, volume 1, pages 135–144, Kyoto, Japan, April 1996.

[17] Song Chong and San-Qi Li. Probabilistic burstiness-curve-based connection control for real-time multimedia services in ATM networks. *IEEE Journal on Selected Areas of Communications*, 15(6):1072–1086, August 1997.

[18] A.K. Choudhury and E.L. Hahne. Dynamic queue length thresholds for shared-memory packet switches. *ACM/IEEE Transactions on Networking*, 6(2):130–140, April 1998.

[19] I. Cidon, A. Gupta, T. Hsiao, A. Khamisy, A. Parekh, R. Rom, and M. Sidi. OPENET: an open and efficient control platform for ATM networks. In *Proceedings of INFOCOM*, San Francisco, USA, March-April 1998.

[20] David Clark. Integrated services in the internet architecture: An overview, June 1994. RFC 1633.

[21] David Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of SIGCOMM'92*, Baltimore, Maryland, August 1992.

[22] David Clark and J. Wroclawski. An approach to service allocation in the internet, July 1997. Internet Draft <draft-clark-diff-svc-alloc-00.txt>.

[23] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *ACM/IEEE Transactions on Networking*, 5(6):835–846, December 1997.

[24] Rene L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transaction of Information Theory*, 37(1):114–131, 1991.

[25] Stephan E. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification, December 1995. RFC 1883.

[26] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proc. ACM SIGCOMM'89, pp 3-12.

[27] N. G. Duffield, Pawan Goyal, Albert Greenberg, K. K. Ramakrishnan, and Jacoubs E. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings of SIGCOMM*, August 1999.

[28] A. Eckberg, D. Luan, and D. Lucantoni. Bandwidth management: a congestion control strategy for broadband networks - characterizing the throughput-burstiness filter. In *Proceedings of the ITC Specialist Seminar*, September 1989.

[29] Anwar I. Elwalid and Debasis Mitra. Effective bandwidth of general markovian traffic and admission control of high speed networks. *IEEE/ACM Transactions on Networking*, 1(3):329–343, June 1993.

[30] Hans Eriksson. MBone: The multicast backbone. *Communications of the ACM*, 37:54–60, August 1994.

[31] Domenico Ferrari. Client requirements for real-time communication services. *IEEE Communications Magazine*, 28(11):65–72, November 1990.

[32] Domenico Ferrari. Design and applications of a delay jitter control scheme for packet-switching internetworks. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 72–83, Heidelberg, Germany, November 1991.

[33] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network support for multimedia: a discussion of the Tenet approach. *Computer Networks and ISDN Systems*, 26:1267–1280, July 1994.

[34] Domenico Ferrari and Dinesh Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[35] Domenico Ferrari and Hui Zhang. Improving utilization for deterministic services in multimedia communication. In *IEEE International Conference on Multimedia Computing and Systems*, pages 295–305, Boston, Massachusetts, May 1994. ACM, Prentice Hall.

[36] Danilo Florissi. The NetScript approach to programmable networks. In *Proceedings of OPENSIG'98 - Workshop on Open Signaling for ATM, Internet and Mobile Networks*, Toronto, October 1998.

[37] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *ACM/IEEE Transactions on Networking*, 1(4):397–413, August 1993.

[38] Sally Floyd and Van Jacobson. Link sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[39] S. Fotedar, M. Gerla, P. Crocetti, and L. Fratta. ATM virtual private networks. *Communications of the ACM*, 38:101–109, February 1995.

[40] Alexander G. Fraser, Chuck R. Kalmanek, A.E. Kaplan, William T. Marshall, and R.C. Restrick. Xunet2: A nationwide testbed in high-speed networking. In *Proceedings of INFOCOM*, pages 582–589, Firense, Italy, May 1992.

[41] Ron Frederick. An Internet posting: Some info on the nv 2.x packet format, December 1992.

[42] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):305–313, April 1991.

[43] G. Gallassi, G. Rigolio, and L. Fratta. ATM: bandwidth assignment and bandwidth enforcing policies. In *Proceedings of IEEE Globecom'89*, November 1989.

[44] Rahul Garg and Xiaoqiang Chen. RRR: Recursive round robin scheduler. In *Proceedings of IEEE Global Telecommunications Conference 1998*, Sydeny, November 1998.

[45] Mark William Garrett. *Contributions Toward Real-Time Services on Packet Switched Networks*. PhD dissertation, Columbia University, 1993.

[46] Mark William Garrett and Walter Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *Proceedings of SIGCOMM*, pages 269–280, London, UK, September 1994.

[47] Leonidas Georgiadis, Roch Guerin, and Abhay Parekh. Optimal multiplexing on a single link: Delay and buffer requirements. In *Proceedings of INFOCOM*, pages 524–532, Toronto, Canada, June 1994.

[48] Richard J. Gibbens and Frank P. Kelly. Network programming methods for loss networks. *IEEE Journal on Selected Areas in Communications, invited paper for special issue on Advances in the Fundamentals of Networking*, 13(7):1189–1198, 1995.

[49] S. Giordano, M. Pagano, R. Pannocchia, and F. Russo. A new call admission control scheme based on the self similar nature of multimedia traffic. In *Proceedings of IEEE International Conference on Communications*, Dallas, Texas, June 1996.

[50] B. Gleeson, A. Lin, J. Heinanen, and G. Armitage. A framework for IP based virtual private networks, September 1998. Internet Draft <draft-gleeson-vpn-framework-00.txt>.

[51] S. Jamaloddin Golestani. A Stop-and-Go queueing framework for congestion management. In *Proceedings of SIGCOMM*, pages 8–18, Philadelphia Pennsylvania, September 1990.

[52] S. Jamaloddin Golestani. A self-clocked fair queuing scheme for broadband applications. In *Proceedings of INFOCOM*, pages 636–646, April 1994.

[53] Pawan Goyal, Albery Greenberg, Chuck Kalmanek, Bill Marshal, Partho P. Mishra, Doug Nortz, and K. K. Ramakrishnan. Integration of call signaling and resource management for IP telephony. *IEEE Network*, 13(3):24–32, May-June 1999.

[54] Roch Guérin, Hamid Ahmadi, and Mahmoud Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas of Communications*, 9(7):968–981, September 1991.

[55] Amit Gupta and Raphael Rom. Improving VC set-up, management and use via bunching. In *Proceedings of the 2nd Workshop on Traffic Management and Synthesis of ATM Networks*, September 1997.

[56] R. Gurenefelder, J. P. Cosmas, S. Manthrope, and A. Odinma-Okafor. Characterization of video codecs as autoregressive moving average processes and related queueing system performance. *IEEE Journal on Selected Areas of Communications*, pages 284–293, 1991.

[57] K. Hamzeh. Ascend tunnel management protocol - ATMP, February 1997. RFC 2107.

[58] D. P. Heyman, A. Tabatabai, and T. V. Lakshman. Statistical analysis and simulation study of video teleconference traffic in ATM networks. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 49–59, 1992.

[59] Robert M. Hinden. IP next generation overview. *CACM*, 39(6):60–71, June 1996.

[60] Jay Hyman, A. Lazar, and C. Pacifi. Real-time scheduling with quality of service constraints. *IEEE Journal on Selected Areas of Communications*, pages 1052–1063, September 1991.

[61] Jay Hyman, Aurel A. Lazar, and Giovanni Pacifici. Joint scheduling and admission control for ATS-based switching nodes. *Computer Communications Review*, 22(4), October 1992. SIGCOMM '92 Symposium.

[62] Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s part 2: Video. ISO/IEC 11172-2, 1993.

[63] Information technology – generic coding of moving pictures and associated audio information: Video. ISO/IEC DIS 13818-2, 1996.

[64] Jeffrey M. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, COM-29(7):954–961, July 1981.

[65] Raj Jain, Arjan Durresi, and Gojko Babic. Throughput fairness index: An explanation. ATM Forum Contribution: AF/99-0045, February 1999.

[66] D. Jamieson, B. Jamoussi, G. Wright, and P. Beaubien. MPLS VPN architecture, August 1998. Internet Draft <draft-jamieson-mpls-vpn-00.txt>.

[67] Laurent Jaussi, Martin Lorang, and Jordi Nelissen. A detailed experimental performance evaluation on TCP over UBR. In *IEEE International Conference on ATM '98*, Colmar, France, June 1998.

[68] Charles R. Kalmanek, Hemant Kanakia, and Srinivasan Keshav. Rate controlled servers for very high-speed networks. In *IEEE Global Telecommunications Conference*, pages 300.3.1 – 300.3.9, San Diego, California, December 1990.

[69] Hemant Kanakia, Partho P. Mishra, and A. R. Reibman. An adaptive congestion control scheme for real-time packet video transport. *Computer Communication Review*, 23:20–31, October 1993.

[70] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing in a space-division packet switch. *IEEE Transactions on Communications*, 32(12), December 1987.

[71] Kenji Kawahara, Kouichirou Kitajima, Tetsuya Takine, and Yuji Oie. Packet loss performance of selective cell discard schemes in ATM networks. *IEEE Journal on Selected Areas of Communications*, 15(5):903–913, June 1997.

[72] Frank P. Kelly. *Loss networks.* University of Cambridge, Cambridge, England, 1989.

[73] Srinivasan Keshav. *An Engineering Approach to Computer Networking.* Addison Wesley, Reading, MA, 1997.

[74] Leonard Kleinrock. *Queueing systems.* John Wiley and Sons, 1975.

[75] Leonard Kleinrock. *Queueing systems, Volume 2: Computer Applications.* John Wiley and Sons, 1976.

[76] Jim Kurose. Open issues and challenges in providing quality of service guarantees in high-speed networks. *Computer Communications Review*, 23(1):6–15, January 1993.

[77] Aurel A. Lazar and Giovanni Pacifici. Control of resources in broadband networks with quality of service guarantees. *IEEE Communication Magazine*, pages 66–73, October 1991.

[78] Aurel A. Lazar and Giovanni Pacifici. Control of resources in broadband networks with quality of service guarantees. *IEEE Communications Magzine*, 29(10):66–73, October 1991.

[79] Aurel A. Lazar and Giovanni Pacifici. Management and control of resources in broadband networks with quality of service guarantees. In *Workshop on Distributed Systems: Operations and Management (DSOM '91)*, Santa Barbara, California, October 1991.

[80] Aurel A. Lazar, Giovanni Pacifici, and Dimitrios E. Pendarakis. Modeling video sources for real time scheduling. *Multimedia Systems*, 1(6):253–266, April 1994.

[81] D. S. Lee, B. Melamed, A. R. Reibman, and B. Sengupta. Analysis of a video multiplexer using TES as a modeling methodology. In *Proceedings of IEEE Global Telecommunications Conference*, pages 131–135, 1991.

[82] D.C. Lee, D.L. Lough, S.F. Midkiff, IV Davis, N.J., et al. The next generation of the Internet: aspects of the Internet protocol version 6. *IEEE Network*, 12(1):28–33, January - February 1998.

[83] W. E. Leland, M. S. Taqqu, Walter Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. *ACM Computer Communications Review*, 25(1):202–213, 1995.

[84] J. Liebeherr, D.E. Wrege, and D. Ferrari. Exact admission control for networks with a bounded delay service. *ACM/IEEE Transactions on Networking*, 4(6):885–901, December 1996.

[85] Dong Lin and Robert Morris. Dynamics of random early detection. In *Proceedings of the ACM SIGCOMM Conference : Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM-97)*, volume 27,4 of *Computer Communication Review*, pages 127–138, New York, September14–18 1997. ACM Press.

[86] K. Lindberger and S. E. Tidblom. Weighted fair queueing, a method to control integrated heterogeneous traffic streams, having different QoS demands. In *Twelfth Nordic Teletraffic Seminar NTS12 (VTT Symposium 154)*, pages 47–58, Finland Technical Research Centre, Finland, August 1995. Finland Technical Research Centre.

[87] S. Sidropoulos M. Katevenis and C. Courcoubeti. Weighted round robin cell multiplexing in a general purpose ATM switch chip. *IEEE Journal on Selected Areas of Communications*, 9:1265–1279, October 1991.

[88] B. Maglaris, D. Anastasiou, P. Sen, G. Karlsson, and Robbins J. Performance models of statisitcal multiplexing in packet video communications. *IEEE Transactions on Communications*, pages 834–844, 1988.

[89] A.S. Maunder and P.S. Min. Investigation of rate control in routing policies for B-ISDN networks. In *International Teletraffic Congress - ITC 15*, June 1997.

[90] B. Melamed, D. Raychaudhuri, B. Sengupta, and J. Zdepski. TES-based traffic modeling for performance evaluation of integrated networks. In *Proceedings of INFOCOM*, pages 75–84, 1992.

[91] Nelu Mihai. Geoplex: An open service platform. In *Proceedings of OPENSIG'98 - Workshop on Open Signaling for ATM, Internet and Mobile Networks*, Toronto, October 1998.

[92] Debasis Mitra, Richard J. Gibbens, and Baosheng D. Huang. State-dependent routing on symmetric loss networks with trunk reservation - I. *IEEE Transactions on Communications*, 41(2):400–411, February 1993.

[93] Debasis Mitra and Ilze Ziedins. Hierarchical virtual partitioning: Algorithms for virtual private networking. In *IEEE Global Telecommunications Conference*, pages 1784–1791, 1997.

[94] Kathleen Nichols et al. A two-bit differentiated services architecture for the internet, November 1997. Internet Draft <draft-nichols-diff-svc-arch-00.txt>.

[95] Kathleen Nichols and others. Differentiated services operation model and definitions, February 1998. Internet Draft <draft-nichols-dsopdef-00.txt>.

[96] Ariel Orda, Giovanni Pacifici, and Dimitrios Pendarakis. An adaptive virtual path allocation policy for broadband networks. In *Proceedings of INFOCOM*, volume 1, pages 329–336, San Francisco, USA, March 1996.

[97] Venkata Padmanabhan and Jeffrey Mogul. Improving HTTP latency. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994.

[98] P. Pancha and M. El Zarki. A study of MPEG coding for VBR video transmission. *IEEE Communication Magazine*, 1994.

[99] S.S. Panwar, T.K. Philips, and M.S. Chen. Golden ratio scheduling for flow control with low buffer requirements. *IEEE Transactions on Communications*, 40(4):765–772, April 1992.

[100] A. K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of INFOCOM*, pages 915–924. IEEE, May 1992.

[101] A. K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control - the multiple node case. In *Proceedings of INFOCOM*, pages 521–530. IEEE, March 1993.

[102] Craig Partridge. Editorial: The end of simple traffic models. *IEEE Netowrks*, 7(5), September 1993.

[103] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of Poisson modeling. In *Proceedings of SIGCOMM*, pages 257–268, London, United Kingdom, August 1994. ACM.

[104] William B. Pennebaker and Joan L Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[105] Erwin P. Rathgeb. Policing of realistic VBR video traffic in a ATM network. *International Journal of Digital and Analog Communication Systems*, 6:213–226, 1993.

[106] Martin Reisslein and Keith W. Ross. Call admission for prerecorded sources with packet loss. *IEEE Journal on Selected Areas of Communications*, 15(6):1167–1180, August 1997.

[107] A. Romanow and Sally Floyd. Dynamics of TCP traffic over ATM networks. *IEEE Journal on Selected Areas of Communications*, 13(4):633–641, May 1995.

[108] C. Partridge S. Shenker and R. Guerin. Specification of guaranteed quality of service, September 1997. RFC 2212.

[109] D. Shah and H. Holzbaur. VPNs: security with an uncommon touch. *Data Communications International*, 26(12):53–63, September 1997.

[110] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements, September 1997. RFC 2215.

[111] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proceedings of SIGCOMM*, pages 231–242, September 1995.

[112] W. Simpson. IP in IP tunneling, October 1995. RFC 1853.

[113] P. Skelly, M. Schwartz, and S. Dixit. A histogram based modeling for video traffic behavior in an ATM multiplexer. *ACM/IEEE Transactions on Networking*, pages 446–459, 1993.

[114] J. Stankovic and K. Ramamritham. *Hard real-time systems.* IEEE Computer Society Press, 1988.

[115] Dimitrios Stiliadis. *Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation.* PhD dissertation, University of California Santa Cruz, June 1996.

[116] Dimitrios Stiliadis and Anujan Varma. Design and analysis of frame-based fair queuing: A new traffic scheduling algorithm for packet switched networks. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 104–115, New York, May 1996.

[117] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. In *Proceedings of INFOCOM*, pages 111–119, San Fransisco, California, March 1996.

[118] Dimitrios Stiliadis and Anujan Varma. Efficient fair queueing algorithms for packet-switched networks. *ACM/IEEE Transactions on Networking*, 6(2):164–174, April 1998.

[119] Dimitrios Stiliadis and Anujan Varma. Rate-proportional servers: a design methodology for fair queueing algorithms. *ACM/IEEE Transactions on Networking*, 6(2):164–174, April 1998.

[120] I. Stoica, Hui Zhang, and T. S. Eugene Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. *Computer Communications Review*, 27(4):249–262, October 1997. Also appeared in ACM SIGCOMM'97.

[121] Jonathan Turner. Maintaining high throughput during overload in ATM switches. In *Proceedings of INFOCOM*, San Fransisco, California, March 1996.

[122] Anujan Varma and Dimitrios Stiliadis. Hardware implementation of fair queueing algorithms for asynchronous transfer mode networks. *IEEE Communications Magzine*, pages 54–68, December 1997.

[123] Dinesh Verma. *Guaranteed Performance Communication in High Speed Networks*. PhD dissertation, University of California at Berkeley, November 1991. also, Report No. UCB/CSD 91/663, Computer Sciences Division, UC Berkeley, November 1991.

[124] John Vicente. Genesis: Toward programmable virtual networking. In *Proceedings of OPENSIG'98 - Workshop on Open Signaling for ATM, Internet and Mobile Networks*, Toronto, October 1998.

[125] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):31–44, April 1991.

[126] Ronald Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall, 1989.

[127] J. Wroclawski. Specification of the controlled-load network element service, September 1997. RFC 2211.

[128] Hui Zhang. *Service desciplines for packet-switching integrated-services*. PhD dissertation, University of California at Berkeley, 1993.

[129] Hui Zhang and Jon C. R. Bennett. Hierarchical packet fair queueing algorithms. In *Proceedings of ACM SIGCOMM*, pages 143–156. ACM, September 1997.

[130] Hui Zhang and Domenico Ferrari. Rate-controlled static-priority queueing. In *Proceedings of INFOCOM*, 1993.

[131] Lixia Zhang. Virtual clock : A new traffic control algorithm for packet switching networks. *ACM Transactions on Computer Systems*, 9:101–124, May 1991.

[132] Lixia Zhang, S. Deering, D. Estrin, S. Shenker, et al. RSVP: A new resource reservation protocol. *IEEE Network*, 7(5):8–18, September 1993.

[133] Zhi-Li Zhang and Jim Kurose. Smoothing, statistical multiplexing, and call admission control for stored video. *IEEE Journal on Selected Areas of Communications*, 15(6):1148–1166, August 1997.

Publications related to this thesis

1. Rahul Garg and Huzur Saran. Fair bandwidth sharing among virtual networks: A capacity resizing approach. In *Proceedings of INFOCOM'2000*, Tel-Aviv, Israel, March 2000.

2. Rahul Garg and Huzur Saran. Scheduling algorithms for bounded delay service in virtual networks. In *Proceedings of IEEE Global Telecommunications Conference, GLOBECOM'99*, Rio de Janeiro, Brazil, December 1999.

3. Rahul Garg and Xiaoqiang Chen. RRR: Recursive round robin scheduler. Computer Networks. 31:1951-1966, 1999. Also in *Proceedings of IEEE Global Telecommunications Conference, GLOBECOM'98*, Sydney, November 1998.

4. Rahul Garg. Characterization of video traffic. Accepted for presentation at *Applied Telecommunication Symposium 1999*, San Diego, USA, 1999.

5. Rahul Garg. Performance evaluation of deterministic guarantees. In *Proceedings of Fifth International Conference on Advanced Computing*, Chennai, India, December 1997.

BIO-DATA

Rahul Garg (born 1n 1972) received B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Delhi in 1993. He completed his MS from University of California at Berkeley, USA in 1995. He worked with Network Program Inc., as a technical lead for an ATM signaling project. He joined Ph.D. at IIT-Delhi in December 1996. He has received several prizes, scholarships and awards during his studies and employment. He has visited Bell Labs and Sun Microsystems Labs during his MS and Ph.D. He has worked and collaborated with researchers in different areas. His areas of interest are broadband Integrated Services networks, traffic management, quality of service, pricing and bandwidth allocation, protocols, algorithms, MPEG video processing and auction theory.