ABSTRACT

BENGERI, SUDHINDRA S. Differentiated Services Support for the Helios Optical Access Network Testbed. (Under the direction of Dr. George Rouskas.)

We consider the problem of scheduling a differentiated mix of (IP DiffServ type) traffic in a broadcast single hop WDM network. Tunability is provided only at one end, namely at the transmitter. Our objective is to design transmission schedules to schedule a mix of guaranteed service and best-effort traffic, by allocating the excess bandwidth to best-effort traffic in a *max-min fair* manner. We consider scheduling algorithms for both systems with negligible transceiver tuning latency and systems with non-negligible transceiver tuning latency. We map the optimal preemptive *Open-Shop* scheduling algorithm to schedule packets in a system with negligible transceiver tuning latency and use the nonpreemtive *OSTL* scheduling algorithm for systems with non-negligible transceiver tuning latency. We propose mechanisms for the allocation of slots to best effort traffic for the preemptive and non-preemptive scheduling algorithms. We present the results of extensive simulation study that evaluate the performance of the best effort allocation schemes in terms of channel throughput and schedule lengths. We identify the effect of schedule lengths and percentage reservations for guaranteed traffic on scheduling delay, and develop heuristics that effectively decouple the effect of schedule lengths on scheduling delay. We then present the architecture of the highly extensible WDM simulator component we implemented over *ns-2*. The component can be used to study the performance of collision-free scheduling algorithms and queuing mechanisms on a TT-FR broadcast single hop WDM network. Finally, we use this simulator component to study the effects of different system parameters, such as slot demands and reservations on scheduling delay, and delay jitter.

DIFFERENTIATED SERVICES SUPPORT FOR THE HELIOS OPTICAL ACCESS NETWORK TESTBED

by

Sudhindra Suresh Bengeri

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Networking

Raleigh

2001

APPROVED BY:

Chair of Advisory Committee

BIOGRAPHY

Sudhindra Suresh Bengeri was born in Belgaum, India on May 10th, 1976. He received his bachelors degree in Computer Science and Engineering from the department of Computer Science at Gogte Institute of Technology, Belgaum, India in 1993. He was with IBM Global Services India from 1997 to 1999 where he was part of the systems software group. He joined the department of Computer Science at the North Carolina State University, Raleigh, NC in fall of 1999. He is currently working towards completion of his master's degree in Computer Networking. He is a member of *Upsilon Pi Epsilon* (UPE), an International honor society for the computer sciences.

ACKNOWLEDGEMENTS

I would like to thank my parents for everything that they have done for my upbringing and encouraging me to pursue higher education. My fiancee, Shilpa, has been a constant source of motivation from the time I took *GRE*, through the application process and during my masters. I thank for having stood by me in every decision I have made.

I would like to thank Dr. George N. Rouskas for having given me the opportunity for conducting research in the interesting field of WDM access networks, under his able guidance. This thesis would not have be possible without his continual support and advice.

I would like to thank Dr. Harry Perros and Dr. Fornaro for serving on my thesis committee. I would also like to thank Dr. Carla Savage for her suggestions for efficient implementation of the *Open-Shop* scheduling algorithm.

I thank Vikram Pendharkar for taking time to review my thesis. Last, but not the least I would like to take this opportunity to acknowledge my sisters and Shilpa's parents for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Broadcast-and-Select Single-Hop WDM Optical Networks

Technological advances have dramatically increased electronic processing speeds, however so has the transmission capacity of optical systems. Inevitably, network capacity will always be limited by the electronic bottleneck. Wavelength division multiplexing (WDM) eases this opto-electronic speed mismatch by partitioning the enormous optical bandwidth into multiple channels. Each channel carries traffic at data-rate of the interface electronics, possibly at peak electronic speed. By allowing multiple WDM channels to coexist on a single fiber, one can tap into the huge fiber bandwidth, with the corresponding challenge being design and development of appropriate network architectures, protocols and algorithms. Taking advantage of the emerging optical networking technologies [9], many researchers, both in academia and industry, have contributed significantly to the theoretical and practical aspects of realization of WDM networks. WDM transmission systems with transmission capacity exceeding Tbps have been demonstrated; and systems supporting hundreds of Gbps are becoming commercially available. A number of experimental prototypes have been (Monet [5], Rainbow-II [4]) and are currently being developed (Helios [2], ONRAMP [3], Hornet, SuperNet, NGI [8]).

WDM network architecture of interest in this thesis is the Broadcast-and-select single-hop architecture [10], which is all-optical in nature, i.e., any information transmitted into the medium remains in the optical form until it reaches its destination. There are no opto-electronic conversions and no buffering/queueing inside the network.

The traditional application for WDM systems is the transport of SONET/SDH signals generated by Add Drop Multiplexers (ADMs). Due to the increase in data traffic a large portion of this traffic is carried over IP, ATM or Frame relay. SONET/SDH were developed for supporting voice traffic, and are not ideally suited to meet the demands of data traffic. Hence a lot of research is directed towards transmitting packets directly over WDM, from design of such networks [11] to efficient packet scheduling.

1.2 Differentiated Services

ATM was designed with a view to provide services such as teleconferencing, video on demand(VOD), in addition to voice services, all over a single network. Different classes of service were designed to support different packet loss and delay requirements. In the traditional IP networks, however, all user packets compete equally for the network resources. With the usage and popularity of IP networks, a significant burden is placed on the limited network resources, such as bandwidth and buffer space, resulting in heavy congestion. Such congestion does not encourage adoption of IP networks as transport mechanisms for real-time and mission critical applications. Differentiated Services [13] or Diffserv, is an IP QoS architecture based on packet-marking that allows packets to be prioritized according to user requirements (the reader is referred to [17] for a framework of current IP QoS).

Differentiated services enhancements to the Internet protocol are intended to enable scalable service discrimination in the Internet without the need for per-flow state and signaling at every hop. The Differentiated services architecture [13] achieves scalability by implementing complex classification and conditioning functions only at network boundary nodes, and by applying per-hop behaviors to aggregates of traffic which have been appropriately marked using the DS field. The previous statement uses the terms *per-hop behavior* (PHB) and *DS field* [12] which are the two building blocks of this architecture, using which services can be built. PHB denotes a combination of forwarding, classification, scheduling and drop behaviors at each hop. PHB is primarily a description of desired behavior on a relatively high abstraction level, and should allow the construction of predictable services [16]. The DS field supersedes the existing definitions of the IPv4 TOS octet and the IPv6 Traffic Class octet. Six bits of the DS field are used as a codepoint (DSCP) to select the per-hop behavior a packet experiences at each node.

The definitions of PHBs is a critical part of the work of the Diffserv working group. An Expedited Forwarding (EF) PHB [14] can be used to build a low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS domains. Assured Forwarding (AF) PHB [15] group is a means for offering different levels of forwarding assurances for IP packets. Four AF classes are defined, where each AF class is in each DS node allocated a certain amount of forwarding resources (buffer space and bandwidth). Within each AF class IP packets are marked with one of three possible drop precedence values. In case of congestion, the drop precedence of a packet determines the relative importance of the packet within the AF class.

1.3 Motivation

To honor ATM CoS/IP QoS parameters appropriate end-to-end mechanisms need to be devised, that include mechanisms in the backbone and access networks. Until recently research in single-hop packet scheduling algorithms primarily focused on minimizing average delay and increasing network throughput [28]. These algorithms could not guarantee Quality-of-Service parameters such as bandwidth guarantee, cell transfer delay and cell delay variation. Scheduling algorithms should guarantee bandwidth or delay requirements of traffic classes and should be able to distribute excess bandwidth to best effort traffic in a *max-min fair* manner. At the same time, scheduling algorithms should make efficient utilization of system resources and yield high channel throughput.

The primary motivation for this thesis is to provide the basis for Differentiated Services support for the Helios [2] testbed. Helios is a DARPA ITO sponsored research project, and is a part of DARPA's NGI program [1]. In this thesis we explore scheduling algorithms that can be used to schedule IP Diffserv type traffic. We desgin mechanisms for efficient distribution of excess bandwidth to best-effort traffic. We also identify parameters that affect the scheduling delay and propose heuristics that reduce average delay and delay jitter.

1.4 Thesis Organization

The thesis is organized as follows. In Chapter 2 we identify the parameters affecting the design of packet scheduling algorithms that provide either bandwidth or delay guarantees. We then, survey the algorithms proposed in literature and classify them. In Chapter 3 we describe the system model on which this thesis is based, and state the problem that this thesis addresses. In Chapter 4 we discuss scheduling algorithms which we use to schedule a mix of guaranteed and best effort traffic. We then propose simple mechanisms to allocate slots to best effort traffic and provide simulation results related to the channel throughput and schedule lengths. In Chapter 5 we present heuristics that reduce variation in delay experienced by packets being scheduled in bursts by WDM scheduling algorithms. In Chapter 6 we present the architecture and implementation details of the WDM simulator component we implemented on *ns-2* and present some delay-related performance results in Chapter 7. Finally we summarize our work in Chapter 8 and provide directions for future research.

Chapter 2

Background and Related Work

In this chapter, we survey some of the approaches and techniques proposed in recent publications for supporting QoS guarantees and delay-contrained communication on local lightwave networks. We first identify the parameters affecting the design of packet scheduling algorithms that provide either bandwidth or delay guarantees. Next, we classify the protocols and scheduling algorithms proposed in the literature. We then explain the important characteristics of the surveyed algorithms, and discuss their advantages and disadvantages.

2.1 Algorithm design parameters

In a broadcst WDM network, there are *three* types of resources - wavelenghts, transmitters and receivers. To transmit a packet users contend for and obtain each of these three resources. Two packets simultaneously transmitted on the same wavelength will result in a *collision.* A transmitter having permission to transmit on two or more wavelengths in a given timeslot, results in a *transmitter conflict.* And two or more packets destined to the same receiver each on a different wavelength, results in a *destination conflict.* In general, collisions of any sort decrease throughput, hence packet loss due to collisions must either be avoided or minimized. Scheduling algorithms provide the necessary coordination between the transmitters and receivers for successful packet transmission.

A scheduling algorithm should allocate sufficient bandwidth to each service class over some interval of time; real-time services should receive a bounded delay and non real-time services should not starve for bandwidth; any excess bandwidth must be distributed fairly to the best-effort service class. Further, a scheduling algorithm should make efficient use of system resources, and yield a high channel throughput.

The design of the scheduling algorithms is strongly dependent on the underlying assumptions regarding the architecture and parameters of the broadcast WDM network. Differences in issues such as tunability characteristics, and signaling method employed result in quite disparate strategies. For the rest of this chapter we take a closer look at the issues which can affect the design of the algorithms to schedule guaranteed traffic. (These parameters are later used for algorithm classification.)

Tunability characteristics The tunability characteristics of a transceiver have a significant impact on the design of scheduling algorithms. Systems that assume tunability at both the transmitter and the receiver are most flexible, but require additional coordination overhead. Also tunability at both the transmitter and the receiver comes at an additional cost. Hence most of the scheduling algorithms consider a tunable transmitter (or receiver) and fixed receiver (or transmitter) configuration. But these systems need to address the problem of assignment of home channels to the fixed receivers (or transmitters), a load balancing problem. In practice, busier nodes (e.g., web servers, routers to other networks) might have a better performance if the scheduling algorithm has the capability to accomodate more than one transceiver at a node [33, 38]. The presence of additional tunable components increases the number of packets a node can transmit/receive in a given timeslot, hence increasing the network throughput. Multiple transceivers could also overlap the transmitter/receiver tuning with transmission to hide long transceiver tuning time.

Tuning latency is the time required for the tunable components to tune from one wavelength to another. While research and development in optical device technology has considerably reduced the tuning latency of componenents, recent advances have also increased the electronic transmission speeds. Hence depending on the packet size and the data transmission rate, the tuning latency could still be on the order of several packet transmission times. If faster components or larger packet transmission times are assumed, the tuning latency could be considered negligibile compared to the length of a timeslot [31, 35, 33, 41, 43], and can be accounted for by having appropriate guard bands around the packet transmission time within each slot. In this case simpler non-preemptive algorithms could be employed that yield an optimal schedule length. If the tuning latency is large, then including it in the slot time increases average packet delay and reduces the network throughput, e.g., if the tuning time is equal to the packet transmission time and is considered part of the slot time, effectively only 50% of the available bandwidth is utilized. Algorithms that consider arbitrary tuning latencies are quite complex, and we did not come across a algorithm that solves this problem explicitly ([38] requires only moderate tuning speeds, however the algorithm does not operate in the time-slotted fashion).

Bandwidth allocation The media access control protocols proposed for single hop WDMA based optical networks can be broadly classified into *reservation based*, *preallocation based* and a form of *node polling*.

Preallocation based techniques pre-assign the channels to the nodes either for data transmission or for data reception. The channel which a node uses to transmit or receive data is prespecified using time division multiple access (TDMA) or its variants [36, 41]. The advantages of this technique are no control overhead, low per-packet processing overhead, and no collisions during control/data packet transmission (and hence higher throughput can be achieved). However, it cannot accomodate dynamic allocation of resources based on setup/clear requests.

Reservation based techniques could either employ *in-band* or *out-of-band* signaling. Out-of-band signaling techniques designate one or more wavelength channels as the control channels and use them to reserve/coordinate access on the remaining data channels for data transmission [31, 32, 35, 33, 37, 43]. The control channel could also be used for other functions including, but not limited to, new node registration/initialization, network management & monitoring, and global clock synchronization. Separate fixed-tuned transceiver(s) are employed for communication over the control channel. The control channel(s) and the transceiver(s) cannot be used for data transmission and attribute towards the *control overhead*. In case of in-band signaling, control packets are sent over the same channels used for data [29]. In-band signaling has the advantages of significantly reduced control overhead, in that control channel and its associated transceivers are not required, and it enjoys all the other advantages of a reservation-based scheme.

From the *Node polling* class, token passing is a common scheme [38, 39]. Token passing protocols are characterized by algorithmic simplicity, collisionless nature, and ease of modification. However such schemes, in general, do not yield high throughput.

Centralized vs. Distributed architecture In a *distributed architecture* each node runs an identical copy of the scheduling algorithm and the reservation scheme distributes the control information, queue size and destination, required to all the nodes [31, 37, 38, 39, 41, 43]. The distributed architecture is scalable and does not have a single point of failure. However, it assumes that all the nodes in the network have the required processing power to compute the schedule. In the *centralized architecture*, all the nodes share queue length and other traffic control information with a central scheduler. The scheduler computes the schedule periodically and then distributes it to all the nodes in the network [32, 35, 33, 36, 40]. The scheduler communicates with the nodes via a dedicated control channel, and vice versa, hence two control channels are required. Because of the control channel delays, a newly arrived packet must wait until its presence is reported to the scheduler, the scheduler computes the new schedule and disseminates the schedule to all the nodes. The centralized scheme suits well for networks with smaller propagation delays, whereas the distributed algorithm is more suitable for networks with larger propagation delays.

Scheduling slot-by-slot vs. Scheduling a cycle Schedulers which schedule packets on a slot-by-slot basis, run the scheduling algorithm every slot time [32, 35, 43]. Hence simple algorithms need to be designed to keep the running time of each iteration low. This normally results in a local maximum and such algorithms may not achieve optimal performance. However, such scheduling algorithms yield schedules that reduce average packet delay, i.e., better response times. Also when such schedulers are used in the centralized architecture, they operate in a pipelined fashion to mask the propagation delay [35]. Schedulers which schedule packets in a cycle, collect queue length information at each of the nodes and execute the scheduling algorithm that tries to reduce the schedule length and the average packet delay [31, 36, 37, 41]. The running time of such algorithms is considerably high, but may yield schedules that have close to optimal schedule lengths.

Unit of fairness Bandwidth reservations could be granted to individual message streams (or virtual circuits) or to groups of packets belonging to a particular traffic class (i.e., class-based QoS guarantees). In order to satisfy per-VC bandwidth and delay guarantees, state information needs to be stored on a per-VC basis, and also each VC needs to have its own queue at its source node [35, 33, 36, 37]. However, it facilitates isolation of VCs from each other. Algorithms based on VCs [34] are better suited to carry ATM-style traffic. Algorithms that provide real-time guarantees are based on individual message streams [36, 37].

Algorithms that provide class-based QoS guarantees schedule aggregated message streams, to satisfy the bandwidth and delay requirements of each class [31, 32, 41, 43]. In order to support QoS, r different queues for each destination are maintained at each node, where each queue corresponds to a different QoS class. Such algorithms are better suited to provide IP DiffServ-like QoS.

Need of higher level module for policing traffic The problem of providing guaranteed performance service to application streams (VCs/QoS classes) is tied to three key issues: 1) admission control of the application streams; 2) characterization of the application streams and policing the traffic; and 3) efficient scheduling algorithms to manage the messages' transmissions and multiplexing. The design of the scheduling algorithm could assume the presence of any combination of admission control, traffic policers and shapers. The scheduling algorithms proposed in [35, 34, 33] require only admission control to test schedulability of the admitted virtual circuits. The algorithm services VCs such that every user gets the same fair share or gets their assigned weighted share, thus obviating the need of a policer and a shaper. However, most scheduling algorithms also assume the presence of a meter that measures incoming traffic and matches it against the traffic profile set up during admission control. The output of the meter then serves as an input to the *markers, policers* and *shapers.*

2.2 Scheduling Algorithms

Scheduling algorithms that provide QoS guarantees can be broadly classified as those that provide bandwidth guarantees and those that specifically deal with delay-constrained traffic.

1. **Bandwidth guarantee.** The algorithms providing bandwidth guarantees are normally formulated as *Matrix clearing problems.* The traffic pattern is formulated as a traffic matrix, whose (i,j) entry represents the traffic load from node i to node j. This traffic load may be considered actual transmission requests or queue lengths or desired transimission rates. The problem then becomes producing schedules to clear the matrix - after each timeslot, some traffic matrix entries are decreased corresponding to transmissions in that timeslot, and this proceeds timeslot by timeslot until the matrix contains all zeros, i.e., it is cleared. The research in these algorithms focuses

Schedule Computation	Bandwidth Allocation	Unit of fairness	Bandwidth Guarantee	
			Reference	Node Structure
Centralized	Reservation	per VC	[33]	CC-TT*FT-TR*FR
			[35]	CC-TTFT-TRFR
		QoS class	[32]	TT-FR/FT-TR

Table 2.1: Classification of algorithms providing bandwidth guarantee.

Schedule Computation	Bandwidth Allocation	Unit of fairness	Timing guarantee	
			Reference	Node Structure
Centralized	Preallocation	Message streams	[36]	TT-FR
Distributed	Preallocation	QoS class	[41]	FT-TR
	Reservation	Individual messages	[30]	CC-TTFT-TRFR
		Message streams	[31, 37]	CC-TTFT-TRFR
	Token Passing	Traffic class	[39]	FT^C-FR^C
		Message streams	[38]	CC-TT*FT-FR*FR

Table 2.2: Classification of algorithms providing timing guarantee.

on minmizing the average delay, reducing the call blocking probability, increasing the user accessible bandwidth, or, increasing the network throughput [31, 32, 35, 34, 33].

2. **Delay guarantee.** These algorithms support delay-constrained communication for messages with delivery deadlines for embedded real-time systems or for interactive distributed services. The algorithms in this class schedule packet transmissions with the objective of meeting message deadlines. Some of the algorithms provide deterministic guarantees [36, 37, 41] and others are mainly targeted for soft real-time applications [38, 39, 30].

Both approaches implicitly guarantee the other factor to some extent, for instance, algorithms that guarantee bandwidth requirements for an application stream do it over an interval of time hence providing soft-real-time guarantees. Also different classes of service could be designed with each class scheduled such that each class experiences different delay and bandwidth characteristics. Similarly algorithms designed to provide delay guarantees indirectly also provide bandwidth guarantees by allocating a given number of slots, to an application stream, every period. The tables 2.2 and 2.2 classify the algorithms being surveyed in this paper on some of the criteria discussed in section 2.1 and the above discussed algorithm classification.

2.2.1 Bandwidth Guarantee

[33, 35] form the bulk of work that has been carried out in the field of scheduling algorithms that provide bandwidth guarantees. All the algorithms proposed follow the greedy approach of algorithm design, and hence are easy to implement but result only in local maximum.

[33] proposes a centralized scheduling mechanism that provides a minimum bandwidth guarantee plus best-effort fair access to excess bandwidth. [33] considers a network of N nodes interconnected by a passive star coupler that supports m wavelengths. Each node can have T_i tunable transmitter and R_i tunable receivers. And a pair of transceivers fixed-tuned to the control channel (a CC-TT^{T_i}FT-TR^{R_i}FR configuration). Data are transmitted in fixed-size cells, where one cell can be sent in one timeslot and tuning latency is assumed to be negligible compared to the length of a timeslot. All the nodes periodically report their VCs' queue lengths to the scheduler.

[33] defines a set of VC transmission rates to be *max-min fair* if and only if every VC has one (or more) *bottleneck resource*. [33] proposes a basic scheduling algorithm that achieves *max-min fair* allocation of resources without any minimum bandwidth guarantees and then extends it to handle minimum bandwidth guarantees. The *basic scheduling algorithm* keeps track of how many cells each VC has sent so far, in a state variable called the VC's *Usage*. During each timeslot the algorithm considers each backlogged VC in increasing *Usage* order, and the VC is assigned the timeslot if it does not violate the transmitter, receiver and wavelength constraints. Whenever a timeslot is assigned, that VC's *Usage* is incremented by one. Each VC is considered only once, at which point it is either added to the transmitting set or skipped, hence the resulting algorithm is simple and fast. The authors also suggest the data structures that facilitate efficient implementation of the basic scheduling algorithm.

[33] assumes that the guaranteed rates of VCs, and the resources used to support

those transmissions are exempt from fairness consideration. The excess resources are shared in a max-min fair manner. The *guaranteed bandwidth* (GBW) algorithm maintains a *Credit* variable with each VC. If a VC has a guaranteed rate of g cells/time slot, then its *Credit* variable is incremented by g every timeslot. VCs are sorted by amount of excess bandwidth it has used beyond its GBW ($ExcessUsage = Usage - \lfloor Credit \rfloor$). The algorithm tries to schedule VCs in a greedy fashion in increasing $ExcessUsage$ order. [33] proves that the algorithm respects 50% bandwidth reservation.

[35] proposes a class of algorithms called *maximal weighted matching* algorithms to choose VCs for transmission based on their bandwidth reservations. [35] considers a similar system model as described above ([33]), except that each node is assumed to have only one pair of tunable transceivers and a pair of fixed tuned transceiver (a CC-TTFT-TRFR configuration). The WDM broadcast network is modeled as a bipartite graph, with the set of source nodes U and the set of destination nodes V forming the two partite sets and every edge $e \in E$ represents a VC from some $u \in U$ to some $v \in V$. An *m-matching* is a matching with m or fewer edges. An m-matching for the bipartite graph representation satisfies the transmitter, receiver and wavelength constraints. A numeric *weight* $w(e)$ is associated with each edge e, which represents the priority assigned to the corresponding VC. A transmission schedule is then obtained by choosing a *maximal weighted m-matching* every timeslot. [35] has proposed a simple algorithm called the central queue (CQ) algorithm for computing *maximal weighted m-matching.* The algorithm considers edges in decreasing order of weights, selecting edges that form an m-matching. [35] then proposes different weight assignment schemes and their variations.

In the basic *Credit-Weighted Algorithm* (CWA) each VC is given g_f (its bandwidth reservation in cells/timeslot) credits each timeslot. The total amount of credits accumulated by a VC up to time t represent its reserved share up to that time. When a VC transmits a cell its credit account is decremented by one. ($C_f(t)$ denotes the, not yet spent, credits of a VC at time t.) The CWA uses $C_f(t)$ as edge weights. The CWA assigns edge weights only to backlogged VCs with positive $C_f(t)$. Hence the CWA is not "work-conserving".

The CWA cannot bind credits for bursty traffic, in that, idle VCs accumulate credits and hog resources when they become backlogged again. Such behavior is undesirable since it affects the cell delay and delay jitter of other well-behaving VCs. The *Bucket-Credit Weighted Algorithm* (BCWA) associates a bucket size parameter, B_f, with each VC. An idle VC can accumulate a maximum of B_f credits before it becomes backlogged again. The bucket size parameter hence restricts the amount of resources bursty sources can acquire after a *silent* period. Thus reducing its effect on other well-behaved VCs.

The BCWA works well for bursty traffic, its main disadvantage is that VCs may lose credits and therefore deviate from the "ideal" throughput guarantee. The *Validated Queue Algorithm* (VQA) uses the quantity $VQ_f(t) = min(C_f(t), Q_f(t))$ as edge weights, where $Q_f(t)$ denotes the queue length of VC f at time t. The weights $VQ_f(t)$, called the *validated queue lengths*, count the number of queued cells that are eligible for transmission. By definition, $VQ_f(t)$ does not allow idle VCs to accumulate more edge weight, hence reducing the hogging behavior of bursty traffic without the use of buckets. Also the $VQ_f(t)$ term encapsulates a credit term and a queue length term, hence any bound on $VQ_f(t)$ simultaneously acts as a credit bound for overloading VCs, and a *queue length* bound on underloading VCs.

The CWA, BCWA and VQA algorithms allocate VCs to timeslots according to their bandwidth reservations, and since best-effort VCs have a $g_f = 0$ such VCs never get scheduled. Hence explicit mechanisms have been proposed for fair sharing of unreserved bandwidth.

The *Two Phase Usage Weighted Algorithm* (UWA) operates in two phases. In the first phase, the algorithm runs the CWA/BCWA/VQA to produce a matching X. If the $|X| < m$ the algorithm runs the basic scheduling algorithm, presented in [33] (discussed above), to fill up the transmission schedule by assigning timeslots to best-effort VCs.

The *Usage-Credit Weighted Algorithm* (UCWA) combines (B)CWA and UWA into a single pass algorithm. The UCWA assigns weights according to the difference $D_f = C_f - U_f$, the difference in the credit accumulated by the VC and its usage of the excess bandwidth. The UCWA is a "work-conserving" version of (B)CWA, in that, it considers edges with nonpositive weights (which represent VCs that have been assigned bandwidth in excess of their guaranteed bandwidth or VCs with no guaranteed bandwidth) for transmission.

The basic scheduling algorithm and the GBW algorithm proposed in [33] can accomodate variable number of transmitters and receivers at a node. However [33] does not give a specified delay bound. [35] proves that the CWA and BCWA support 50% of bandwidth reservations, i.e., if the total amount of bandwidth reservation is less than 50% then all the VCs are guaranteed to receive their requested bandwidth minus a constant credit bound (i.e., $C_f(t) < B_c$ for all f). [35] also provide a delay bound which is a function of the constant credit bound.

2.2.2 Delay Guarantee

For the rest of this section we discuss the various approaches proposed in literature to provide timing guarantees for delay constrained communication.

Preallocation-based Algorithms

[36] proposes a preallocation-based algorithm for providing deterministic delay guarantees for message transmission. [36] considers a network of N nodes each equipped with a tunable transmitter and a tunable receiver (a TT-TR configuration), interconnected through a passive star coupler that supports W wavelength channels, and assumes $W \leq N$. [36] uses a message model similar to the (r, T)-smooth traffic model. [36] considers a set of n isochronous message streams, $\{M_i = (C_i, D_i, N_i^s, N_i^d) | 1 \leq i \leq n\}$, where C_i is the maximum number of packets in M_i that can arrive in any time interval of length D_i. D_i is the relative transmission deadline for the messages in M_i. N_i^s and N_i^d are the source and destination nodes for the messages in M_i. The message density of an isochronous stream M_i is defined as $\rho(M_i) = C_i/D_i$, and the total message density of a set of isochronous streams $\mathbf{M}$=$\{M_1, M_2, \ldots, M_n\}$ as $\rho(\mathbf{M}) = \sum_{i=1}^{n} C_i/D_i$ The algorithm finds a slot assignment over the W channels such that each message stream M_i is guaranteed to transmit each of its messages before the message deadline D_i subject to the source/destination conflict constraints. It does so by assigning at least C_i slots to M_i for any time interval of length D_i.

[36] first solves a restricted case in a TT-FR (FT-TR) system in which the message streams from a source node are assumed to be all destined to the same destination node. In case of a TT-FR system all the message streams that are transmitted over a wavelength channel are grouped into a message set $M_{\lambda_c} = \{M_i | \lambda(N_i^d) = \lambda_c\}$. A slot assignment scheme allocates slots on each wavelength channel independently, such that in any time interval of length of D_i slots, at least C_i slots are allocated to $M_i \in M_{\lambda_c}$ on the wavelenght channel λ_c for i and c. Given a set of isochronous message streams, first the deadline constraints are specialized to a set $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$ which consists solely of multiples, i.e., D_i divides D_j for all $i < j$. Message streams are then assigned priorities using the *rate-monotonic* scheduling concept. Each M_i is then assigned C_i slots over the wavelength channel λ_c during each time period $[(j-1).D_i, j.D_i]$, for all $M_i \in M_{\lambda_c}$. The assignment is done by assigning the current slot over the wavelength channel λ_c to the message stream with the highest priority among all the unfulfilled message streams. [36] proves the correctness of

the above assignment procedure, that for $\rho(M_{\lambda_c}) \leq 1$ such a assignment always exists. The final composite slot schedule then consists of all the slot schedules, one for each wavelength channel.

[36] then proposes a general slot assignment scheme for TT-TR systems. The scheme consists of three steps, decomposition of message streams into a set of *message sub-streams*, grouping time slots on each wavelength channel into *sub-channels* to facilitate slot assignment in a well-spaced manner, and finally assignment of sub-channels to sub-streams by a mechanism called *binary splitting*. A message stream M_i is decomposed into a set of message sub-streams $S(M_i)$ defined as $\{M_{ij} = (C_{ij}, D_{ij}, N_i^s, N_i^d) \mid 0 \leq j \leq m_i \; and \; C_{ij} = 1\}$, where

$$C_i/D_i = \textstyle\sum_{j=0}^{m_i} C_{ij}/D_{ij},\; m_i = log_2 D_i,\;\; D_{ij} = 2^j \; and \; C_{ij} = \;0 \; or \; 1,\;\; for \; 0 \leq j \leq m_i.$$

To assign at least C_i slots in any time interval of D_i slots, at least one slot needs to be assigned to these sub-streams M_{ij} in any time window of D_{ij}. A $1/d$ sub-channel is defined to consist of evenly spaced slots, of which any two consecutive ones are separated exactly by d slots. A $1/d$ sub-channel can be further divided into k sub-channels, each with density $1/kd$. A transformed frame is formed such that a sub-channel will consist of consecutive slots in the transformed time frame. A mechanism called *binary splitting* is proposed which assigns each message stream M_i sufficient slots to fulfill its delay requirement, subject to the source/destination conflict constraints. Conceptually, binary splitting recursively splits the set of message sub-streams under consideration into two smaller subsets, M_L and M_R, until each subset consists only of a single sub-stream. It also recursively divides a transformed time frame F' into two sub-frames F_L and F_R. A message sub-stream assigned to M_L (M_R) will be assigned a sub-channel in the left (right) sub-frame F_L (F_R). The scheme then assigns sub-channels to wavelength channels subject to the source/destination conflict constraints. [36] proposes the schedulability condition under which the above scheme is guaranteed to yield a valid slot schedule and also provide the proof of correctness of the binary splitting algorithm. [36] assumes that tuning latency of the transceivers is negligible, and also the algorithm does not benefit by having multiple transceivers at a node. Also being a preallocation-based scheme it cannot accommodate dynamic slot reservation/release requests, the work in [37] addresses this problem.

[41] proposes a real-time protocol, based on *Time Division Multiplexing* (TDM) that serves guarantee-seeking messages and best-effort messages for single destination, mul-

ticast, and broadcast transmission. [41] considers a network of N nodes interconnected through a passive star coupler with C data channels, it is assumed that $N = C$. Each node has a fixed tuned transmitter, and a tunable receiver (a FT-TR configuration). Time is slotted and each slot is equal to packet transmission time plus a gap for time synchronization. [41] adapts the *Time-Deterministic Time and Wavelength Division Multiple Access* (TD-TWDMA) medium access scheme proposed in [45]. By use of TDMA the access to each channel is divided into cycles of time-slots. Each node is assigned certain slots to transmit guarantee-seeking messages. However, in the absence of guarantee-seeking messages these slots are released for transmitting best-effort messages from other nodes (or the same node) according to a predetermined scheme. A simple deterministic distributed slot-allocation algorithm does the slot allocation. To prevent destination conflicts each slot is assigned a specific owner. Each cycle is partitioned into $(N-1)N$ data slots and N control slots. The slot-allocation algorithm is based on a predetermined allocation scheme that can be partly overloaded. Two sets of slot allocations are defined: high-priority (the default) and low-priority. Each node transmits a control slot every cycle, which contains the release message, and a list of its high-priority slots it wishes to release for the next cycle.

Since the slots are allocated in a predetermined manner, each node knows the number of high-priority slots it holds towards each destination. Hence it can decide if the demand of an incoming guaranteed message stream can be met. [41] analyzes the worst-case packet latency for a guaranteed message stream. [41] performs better compared to a static TDM system, however simulation results show that the bandwidth utilization is low.

[40] has considered the problem of scheduling m periodic tasks on n, $n < m$, identical processors. [40] represents the *periodic task scheduling* problem as a *maximum network flow* problem. [40] shows that a feasible max-flow assignment exists, which corresponds to feasible schedule for the task scheduling problem.

Each periodic task $i, i = 1, \ldots, m$, is characterized by its computation time C_i and deadline D_i (also the period of the task), with $0 < C_i < D_i$. [40] assumes C_i and D_i, $i = 1, \ldots, m$, are integers and that time is slotted with slot time equal to one unit of processing time. The density of a task i, $\rho_i = C_i/D_i$, and the total message density of a set of m tasks is defined as $\rho = \sum_{i=1}^{n} C_i/D_i$. [40] defines the schedule period $\mathbf{D}$ as the least common multiple of all task periods: $\mathbf{D} = lcm(D_1, \ldots, D_m)$. [40] represents the periodic task scheduling problem as a maximum flow problem. The construction of the network flow makes it possible to transform a feasible flow with integer arc flows into a schedule in which the *Task* and *Processor* constraints are both satisfied. [40] shows that the maximum flow in the network is integer, and that there exists a feasible flow assignment in which all arc flows are integer. Such a flow assignment corresponds to a feasible schedule for the task scheduling problem. Thus proving that $\rho \leq n$, where n is the number of identical processors, is a sufficient condition for scheduling the m tasks such that no deadlines are ever missed.

The scheme presented above (in [40]) can be mapped to a preallocation-based algorithm for providing deterministic timing guarantees for transmitting message streams over a optical WDM single-hop network. The m periodic tasks correspond to m periodic message streams; the n identical processors would correspond to n wavelength channel with each node having an arrayed transmitter and receiver (a FT^C-FR^C configuration).

Reservation-based Algorithms

The slot allocation problem mentioned in [36] (see section 2.2.2) is static in the sense that the set of message streams is fixed and known at system initialization, and no message streams are admitted or terminated afterwards. [37] proposes a dynamic slot allocation scheme, building on the above defined concepts of *specialization, message stream decomposition,* and *channel decomposition.* [37] considers a network of N nodes interconnected through a passive star coupler than can support $W + 1$ wavelength channels, λ_0 is used as the control channel, and $\lambda_i,\ 1 \leq\ i \leq\ W$ are used as data channels. Each station is equipped with (i) a pair of tunable tranceivers and (ii) a pair of fixed tuned transceivers, both tuned to λ_0 (a CC-TTFT-TRFR configuration). Data transmission in the network operates in a slotted mode, and each data slot time is set to the sum of transceiver tuning time plus packet transmission time. The control channel is also time slotted with each control slot equal to a data slot and divided into N mini-slots (each pre-assigned to a node). A distributed dynamic slot allocation algorithm runs concurrently on every node in the network. [37] splits the functions of the dynamic slot allocation scheme into two daemons: the *dynamic slot manager* daemon (DYNMGR), and the *dynamic slot allocator* daemon (DYNALO), which communicate with each other using two FIFO pipes. The DYNMGR receives and processes call setup/clear requests. Upon arrival of a setup request, DYNMGR first specializes the deadline constraints and if the message can be established without violating the source/destination conflict constraints (or if the request is call clear) DYNMGR relays the setup/clear request to other stations by transmitting in the pre-assigned mini-slot on the control channel λ_0. Upon receipt of a setup request on the control channel the DYNMGRs on all the nodes, decompose the message stream and forward the request to DYNALO. DYNALO determines whether or not the message sub-stream can be established.

During system operation, DYNALO assigns W slots to the existing message sub-streams, one on each data channel, according to the next slot requirement of each of the message sub-streams. The DYNALO assigns at least one slot to each sub-stream within its deadline, thus ensuring that each message stream M_i is assigned at least C_i slots over any time interval of D_i. After assigning the W current slots, DYNALO processes the next setup/clear request, if one exists and there are no pending reservation requests. In case of a setup request, DYNALO attempts to assign each of the sub-streams a sub-channel of appropriate density, subject to the source/destination conflicts, starting with the sub-stream with the largest deadline constraint. If sub-channels with the exact required deadline do not exist the sub-channels are split recursively until they match the required deadline or a conflict is encountered. The splitting attempt is made on all available empty sub-channels in the order of descending deadlines, until either a conflict-free empty sub-channel with the required deadline is generated or all empty sub-channels are investigated but no conflict-free empty sub-channel with required deadline is located. If the request is "clear" the sub-channels assigned to the terminated sub-streams are tagged as empty. If there exists more than one empty sub-channel with the same deadline constraint, the empty sub-channels are merged starting with sub-channels that have the largest deadline constraint, and progress in a backward manner until all empty sub-channels are merged. Two sub-channels of the same deadline constraint d can be combined into one with deadline constraint $d/2$.

[37] proves the correctness of the proposed scheme, and claims that a newly admitted message stream can be immediately set up without any delay if the corresponding setup request does not arrive during a transition period. A transition period, defined as time during which a message with (specialized) deadline constraint D_i is being terminated, lasts no more than $4D_i$ slots.

[30] proposes a set of reservation-based scheduling algorithms for scheduling variable-length aperiodic time-constrained messages to meet their hard or soft real-time constraints. [30] considers the same system model as [37], with the exception that length a control slot is the transmission time of a control packet, which a system design parameter. The set of algorithms proposed use the *First Control Packet First Serve* (FCPFS) algorithm [45] to

assign data channels and transmission time slots to selected messages. The basic idea of FCPFS is to assign a message to a data channel that has the earliest available time slot among all other channels. [30] proposes two priority assignment schemes based on the two parameters associated with individual messages: their relative deadline and the length of the message. *Minimum Laxity First* (MLF) assigns higher priority to messages with smaller relative deadline and *Shortest Job First* (SJF) assigns higher priority to shorter messages.

[30] proposes a set of real-time scheduling algorithms, combining the transmission channel and the time slots assignment (FCPFS) algorithm with the messages priority assignment schemes. First class of schedules, *Frame scheduling*, consider messages that arrive at a node in the FCFS order. Only the messages at the head of each queue are reported in the control packet. After the control packets related to these messages reach all the nodes, either MLF or SJF assign priorities to these messages. Once the order of message transmission is determined, the channel assignment algorithm, FCPFS, assigns a channel and time slots to these messages. The algorithm *Frame Minimum Laxity First* (F-MLF) uses MLF for priority assignment and *Frame Shortest Job First* (F-SJF) uses SJF. Since the messages, at each node, are considered in the FCFS order, messages with smaller relative deadline (or shorter messages) may get blocked.

The next strategy, *Frame-and-Queue scheduling*, maintains priority message queues at each node, with priority assigned by MLF or SJF schemes. The messages with highest priority at each node are reported in the control packet. After the control packets related to these messages reach all the nodes, a sequencing algorithm based on the same priority scheme is applied again to sequence the messages for FCPFS assignment. The algorithm *Frame-and-Queue Minimum Laxity First* (FQ-MLF) uses MLF for priority assignment and *Frame-and-Queue Shortest Job First* (FQ-SJF) uses SJF.

[30] shows that the MLF based algorithms reduce the message loss rate, and hence are well suited for scheduling messages with hard real-time constraints. And the SJF based algorithms reduce the average message delays, and hence are well suited for scheduling messages with soft real-time constraints.

The authors of [31] extend their previous work [30]. [31] proposes an admission control policy, a traffic regulator , and a scheduling algorithm to provide guaranteed deterministic performance service to applications' streams composed of real-time variable length messages. [31] uses the same system model and control channel access scheme as that used by [30]. [31] divides the network service into two levels. The upper level is the flow level at which the admission control and traffic regulator manage and control the application streams. The lower level is the message level at which individual messages are scheduled and transmitted.

The scheduling algorithm handles two issues: message sequencing and channel assignment. [31] proposes an *adaptive round-robin and earliest available time scheduling* (ARR-EATS) algorithm to provide guaranteed deterministic bounded delay service, in conjunction with the proposed admission control and traffic policing schemes. The adaptive round-robin algorithm determines the message transmission sequence. The ARR algorithm differs from the traditional round-robin algorithm in that the ARR algorithm serves the current variable length message completely before switching to the next queue. Since the length of each message is different, the service time for each queue is also different. The EATS technique assigns a data channel and transmission time slots to the selected message. The basic idea of the EATS algorithm is to assign a message to a data channel that has the earliest available time slot among all other channels (similar to the FCPFS algorithm [45]).

Token-passing-based Algorithms

[38] has proposed a distributed adaptive protocol, designed to support multiple classes of soft real-time traffic. [38] considers a network of N nodes interconnected through a passive star coupler with M data channels and one control channel. Each node has a pair of fixed tuned transceivers, tuned to the control channel, one or more fixed tuned receivers and one or more tunable transmitters (a CC-TT*FT-FR*FR configuration). Data channels are not slotted hence the algorithm inherently supports variable length messages. A token on the control channel is used to ensure collision-less transmission and to disseminate the latest status information regarding the token-sending node. The token has a designated receiver; however, every node receives a copy and uses it to update its local status tables. The designated receiver determines which free channels to acquire or which currently held channels to release. The *Priority Index Algorithm* (PIA) is used to compute the node's priority index on each idle data channel. The node can acquire those channels for which it estimates it has the maximum priority index over all nodes. The *Transmitter Scheduling Algorithm* (TSA) then decides which of these eligible idle channels to acquire, this information is then written into the token and passed to the next node, identified by the *Flying Target Algorithm* (FTA).

The PIA assigns an estimated priority index to each channel at each node, which depends on the expected time to serve the packets currently queued at a node (towards a channel) and *additional real-time priority* (ARTP) term that captures the real-time QoS requirements of the stream. The ARTP term is adaptively varied according to whether the fraction of packets missing their deadlines is greater, or less, than a prescribed level. ([38] considers the percentage of packets missing their deadlines as the QoS measure.)

The TSA releases all the currently held channels for which the PIA estimates to have a lower than highest priority index. The TSA orders the available channels on which the node estimates itself to be the highest priority node, in the decreasing order of priority and acquires the top n_t channels (where n_t is the number of tunable transmitters of the node). If there are less than n_t eligible channels, the node get a *free ride*, i.e., transmitters are tuned to idle channels and can transmit data for the duration of time it takes the token to reach the next node. The *free ride* mechanism is intended to utilize bandwidth that would otherwise have been wasted.

Once the TSA acquires/releases channels, the FTA identifies the next node to receive the token. The FTA bases its decision on the following criteria (in order): nodes should not starve for the token for more a specified number of token hops, the token is passed to the node that is estimated to have the highest priority among all the idle channels, if the node currently holding the token is selected as the next node or if there are no free channels the next node is selected in a round robin manner.

As can be observed the token passing mechanism used by [38] differs from the traditional token-ring algorithm, in that (i) all the nodes receive a copy of the token, (ii) the token is passed immediately after transmitter assignment as opposed to the end of packet transmission and (iii) the next node to receive the token depends on the current system status. [38] benefits by the presence of multiple transmitters at a node and the protocol does not require super-fast tuning devices.

[39] extends a simple token-passing scheme to schedule a mix of real-time delay-sensitive and non real-time loss-sensitive traffic. [39] considers a network of N nodes interconnected by a passive star coupler with C channels. Each node has an arrayed transmitter and receiver (a FT^C-FR^C configuration). A multiplexer combines the separate signals from the multiple sources onto a single output fiber, and a demultiplexer separates the wavelengths to individual filters of a node. This enables each node to send or receive messages using any channel. Hence the scheduling scheme only needs to address channel collisions.

This system model also facilitates the use of a floating control channel and even in-band signaling. The protocol operates as follows: if two or more channels are available to the node when it transmits, then data and token packets are sent on separate channels. If, however, a second channel is not available the data packets are piggybacked behind the token packet. The token is passed in a logical sequence over all the nodes using a "round robin" strategy. Each node has access to the network once every *token rotation time* (TRT).

In a static scheduling algorithm real-time packets have a higher access priority at any network condition. This scheme results in the non real-time packets experiencing a higher delay and higher packet loss compared to the real-time packets. [39] proposes a scheme in which the access priorities are reassigned dynamically when the queue length of the non real-time packets reaches an upper limit, γ. This assignment is maintained until the queue length reaches a lower limit, α.

Based on simulation results [39] suggests that a bigger buffer size should be used for non real-time traffic, but with smaller upper threshold for priority shift to obtain improvement in both average delay and blocking probability (a measure of packet loss) for both the traffic types.

The next chapter discusses the system model on which our work is based.

Chapter 3

System Model and Problem Statement

This chapter explains the system model on which this thesis is based. It introduces various traffic matrices and system parameters. We then state the problem that this thesis addresses.

3.1 Network Model

We consider a broadcast and select network topology with N nodes which are interconnected by a passive star coupler (PSC). The PSC supports C channels, $\lambda_1, \ldots, \lambda_C$, (see Figure 3.1). We assume the network to be *wavelength-limited*, hence $N \gg C$. There are no opto-electronic conversions and no buffering/queuing inside the network. Each node is equipped with a transmitter and a receiver. Tunability is provided only at the transmitters[1]. All the transmitters are assumed to have a tuning range that spans all the C wavelengths and it takes equal time to tune from any wavelength to any other wavelength. The receivers are fixed tuned to their assigned home channels[2]. Since we assume a *wavelength-limited* network more than one node is assigned the same home channel. We define R_c as the set of receivers sharing wavelength λ_c:

[1]Slowly tunable receivers could be used to achieve optimal network performance for varying traffic demands.

[2]If slowly tunable receivers are employed this wavelength-to-receiver assignment may change during the reconfiguration phase.

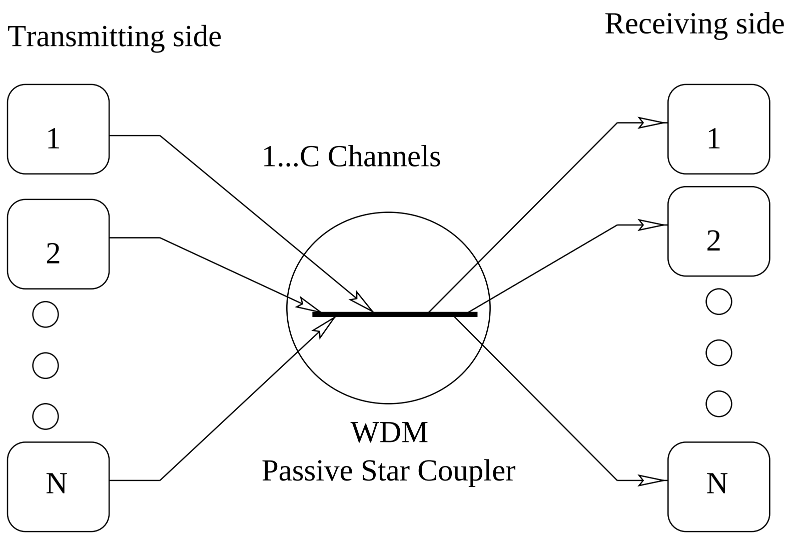

Figure 3.1: A Broadcast-and-select WDM network

$$R_c = \{j|\lambda(j) = \lambda_c\},\ c = 1, \ldots, C \tag{3.1}$$

The system operates in a time slotted manner. Data is transmitted in fixed sized packets, and each slot time is equal to the packet transmission time, plus, possibly the tuning latency. The tuning latency is defined as the time taken by transceivers to tune from one wavelength to another. We consider systems with both negligible and non-negligible transmitter tuning latencies. Each transmitter and each receiver is individually synchronized at slot boundaries (taking into account the propagation delays[3] to the PSC). At any given timeslot, each transmitter can tune to and transmit a packet on one wavelength.

The algorithms we consider require control information about slot reservation requests of each node on each channel. However the algorithms do not dictate the use of any particular pretransmission coordination mechanism. In-band [29] as well as out-of-band mechanisms for control data transmission could be used depending on other system parameters and constraints. That is, schedules can either be computed by a centralized scheduler and distributed to each node or computed individually by each node by running an identical copy of the scheduling algorithm once it receives all the required control information.

We assume that a $N \times N$ traffic demand matrix $\mathbf{D} = [d_{ij}]$. $\mathbf{D}$ is known, with

[3]We do not specify a control protocol hence the propagation delay to the PSC does not affect the algorithm design or analysis, however propagation delay to the PSC needs to be known for slot synchronization.

d_{ij} representing the number of slots to be allocated for transmission of guaranteed traffic from source i to destination j. For example, d_{ij} could be the sum of slots requested for EF traffic and various drop precedences of AF traffic from source i to destination j. Since a transmission on wavelength λ_c is heard by all receivers listening on λ_c, once the receiver-to-channel allocation is completed, the traffic matrix can be collapsed into an $N \times C$ traffic demand matrix $\mathbf{A} = [a_{ic}]$. Element a_{ic} of this collapsed matrix represents the number of slots to be assigned to source i for transmissions on channel λ_c:

$$a_{ic} = \sum_{j \in R_c} d_{ij},\ i = 1, \cdots, N,\ c = 1, \cdots, C \tag{3.2}$$

The scheduling algorithms take as input this collapsed traffic demand matrix to compute a transmission schedule. The transmission schedule may have idle slots during which none of the nodes are permitted to transmit packets on to the channel. Such slots represent wasted bandwidth and result in lower channel throughput. All such slots are ideal candidates for best-effort allocation and could be used for transmitting best-effort traffic. Also admission control algorithms may restrict bandwidth (slot) reservation to a certain percentage (a system parameter) of total available capacity, so that the excess capacity may be used to serve best-effort traffic. We define mechanisms, which exploit characteristics of the scheduling algorithms to compute the best-effort allocation, such that the excess bandwidth is distributed to the best-effort traffic in a *max-min fair* manner. The best-effort allocation is represented by an $N \times C$ matrix $\mathbf{B} = [b_{ic}]$, with b_{ic} representing the number of slots to be allocated to source i for transmission of best-effort traffic on channel λ_c. An $N \times C$ total demand matrix $\mathbf{T} = [t_{ic}]$, element t_{ic} is calculated as the sum of guaranteed and best-effort slots to be allocated to source i for transmission on channel λ_c:

$$t_{ic} = a_{ic} + b_{ic},\ i = 1, \cdots, N,\ c = 1, \cdots, C \tag{3.3}$$

The scheduling algorithms take as input the total demand matrix $\mathbf{T}$ and compute the transmission schedule, let M be the length of the schedule, a $M \times C$ matrix $\mathbf{S} = [s_{sc}]$, where $1 \le s_{sc} \le N$ represents the transmitter that has permission to transmit a packet on channel λ_c during slot s. Any other value of s_{sc}[4] represents an *idle slot.* Each node i should

[4]Each s_{sc} is a scalar term (i.e., refers to only one transmitter), since we consider only scheduling algorithms that yield a collision-less schedule.

have permission to transmit at least t_{ic} slots on channel λ_c over the length of the schedule. If the t_{ic} slots are contiguously allocated for all nodes i on all channels λ_c, the schedule is said to be *non-preemptive*; otherwise it is said to be *preemptive*. Under a *non-preemptive* schedule, each transmitter will tune to each channel exactly once, minimizing the overall time spent for tuning.

In a TT-FR WDM network, packets transmitted by two or more transmitters on the same channel during the same time, result in a *collision*. A transmitter having permission to transmit on two or more wavelengths in a given timeslot, results in a *transmitter conflict*. In order to avoid packet loss due to collisions of any sort, the transmission schedule is subject to the *no-collision constraint* and *transmitter constraint*.

The channel throughput, $\mathcal{S}$, is defined as the average number of successful packets transmitted on the C channels in a time slot. Hence $\mathcal{S} \leq C$. The channel throughput is a significant parameter in the design of scheduling algorithms, since it signifies how well we are utilizing the available channel bandwidth. However, the process of optimizing the network throughput [44] may increase the schedule length, in turn increasing the average delay experienced by packets. Hence the scheduling algorithm that just tries to maximize throughput may not be able to guarantee the bandwidth and delay requirements of the message streams.

3.2 Problem Statement

The length of the transmission schedule cannot be smaller than the number of slots required to satisfy all transmissions on any channel, yielding the *channel bound*. Similarly each transmitter i needs t_{ic} number of slots on each channel λ_c, yielding the *transmitter bound*. Hence the length of the transmission schedule will be greater than or equal to the *lower bound* of the schedule length.

Bounds on Schedule Length

Channel Bound

$$\mathcal{F}_{ch}^{(C)} = \max_{c=1,\ldots,C} \left\{ \sum_{i=1}^{N} t_{ic} \right\} \tag{3.4}$$

Transmitter Bound

$$\mathcal{F}_{tr}^{(C)} = \max_{i=1,\ldots,N} \left\{ \sum_{c=1}^{C} t_{ic} \right\} \tag{3.5}$$

Lower Bound on Schedule length

$$\mathcal{F}_{min} = \max\left\{\mathcal{F}_{tr}^{(1)}, \mathcal{F}_{ch}^{(N)}\right\} \tag{3.6}$$

Objective

A transmission schedule $\mathbf{S}[M \times C]$, with *length of schedule*, M, as close as possible to the lower bound of (3.6), also called the optimal length, and with minimum number of idle slots (that is, high channel throughput, $S \approx C$, without increasing the schedule length), subject to the *no-collision* and *transmitter* constraints.

We consider scheduling algorithms that satisfy the *no-collision* and *transmitter* constraints. Our objective is to schedule a mix of guaranteed traffic and best-effort traffic, by allocating excess (or a specified amount of) bandwidth to best-effort traffic in a *max-min fair* manner. We also identify parameters that affect the *scheduling delays*, and propose heuristics to decouple *delay and delay jitter* from the effect of schedule lengths.

The next chapter explains the scheduling algorithms we have used, we then propose best effort allocation schemes that exploit the characteristics of the scheduling algorithms.

Chapter 4

Scheduling Algorithms

Advances in technology have made possible very fast tunable transmitters. Transceivers with sub-microsecond tuning latency have been demonstrated. Rather than the absolute value of the tuning latency we are interested in its value relative to that of packet transmission time. Let δ denote the *normalized tuning latency*, expressed in units of packet transmission time. The value of δ depends on the electronic data transmission speed, the packet size, and the transceiver tuning time, and can be less than, equal to, or greater than one.

We need algorithms that could make efficient use of resources in systems with $\delta \ll 1$ and also in systems with $\delta \geq 1$ or $\delta \approx 1$. Systems with very high speed tuning transceivers, or larger packet sizes or slower interface date rates, have $\delta \ll 1$. For instance, at interface speeds of 1 Gigabits per second, 1000 byte packets and transceivers with tuning latency 100 nanoseconds, the normalized tuning latency $\delta = 0.0125 \ll 1$. Accordingly, a guard band of δ time units can be added at the beginning of each time slot to allow the transceivers sufficient time to switch between channels, with minimal effects on the channel throughput. For the example above only 1.25% of the total channel bandwidth is wasted for transceiver tuning. For such systems *preemptive* algorithms could be employed to obtain *optimal* performance.

For similar reasons, systems with non-negligible tuning latencies or with smaller sized packets or faster interface electronics, have $\delta \geq 1$ or $\delta \approx 1$. For such systems including the tuning latency of transceivers into every time slot would reduce the channel throughput significantly. For such systems algorithms should make efficient use of system resources by masking tuning time of some transceivers by transmission by other nodes. Also such systems benefit by the use of *non-preemptive* algorithms that reduce the total tuning time by tuning in to a channel only once per cycle.

In this chapter we first map the optimal *preemptive* open-shop scheduling algorithm [26] to schedule packets for single-hop broadcast WDM networks. We then propose a simple and efficient algorithm that can allocate best-effort traffic to obtain 100% channel utilization. We then discuss a *non-preemptive* scheduling algorithm and propose a mechanism that exploits the characteristics of that algorithm to allocate best effort traffic, and to yield near optimal schedule lengths, with minimal number of idle slots.

4.1 Preemptive Open-Shop scheduling algorithm

[26] proposes a *optimal finish time* (OFT) scheduling algorithm for a preemptive *m-processor open shop* and shows that it can be obtained in polynomial time. [26] defines a *shop* as being made up of $m \geq 1$ processors and $n \geq 1$ jobs. Each processor performs a different task and each job i has m tasks (one task to be executed on each processor). Task j of job i requires exactly a_{ij} amount of processing and is to be processed on processor $j, 1 \leq j \leq m$. Further an *open shop* has no restrictions on the order in which the tasks for any job are to be processed. A schedule for a m-shop is a set of m processor schedules, one for each processor in the shop. These m processor schedules must be such that no job is to be processed simultaneously on two or more processors. The finish time is the latest completion time of the individual processor schedules and represents the time at which all tasks have been completed. An *optimal finish time* (OPT) schedule is one which has the least finish time among all schedules.

As can be seen the description of the *open shop* and the requirements of its schedule correspond directly to problem and constraints, respectively, of scheduling packets over a multi-channel single-hop network. The m processors correspond to m channels, the n jobs correspond to the n nodes, and the tasks j of job i corresponds to the slot demand of node i on channel j (an element of the collapsed demand matrix $\mathbf{A}$, see Chapter 3.1). Hence the *optimal preemptive open-shop* scheduling algorithm can be used to schedule packets on a single-hop WDM network. The algorithm has the advantage that it yields optimal schedule lengths hence makes it possible to provision bandwidth as well as specify a bound on the delay parameters. The slot demands could represent bandwidth requirements of aggregated application streams, the algorithm then provides the schedule that can transmit packets according to the bandwidth requirements of guaranteed traffic. We first explain the algorithm briefly and provide an example to show how the algorithm produces optimal schedules. Later, in the next section, we provide an algorithm that can incorporate best-effort traffic into the slot demands without affecting the finish time (schedule length). The algorithm can achieve 100% channel utilization without increasing the schedule length by distributing the excess bandwidth (idle slots) to best-effort traffic.

The scheduling algorithm for a preemptive open-shop [26] makes use of basic concepts from the theory of maximal matchings in bipartite graphs [24]. The m-processor *open shop* can be represented as a bipartite graph with the n jobs, and m processors forming the two partite sets. The tasks form the edge set, with the processing times being the weight of each such edge. Given a set of n jobs with task times $a_{ij}, 1 \leq i \leq n$ and , for a m-processor open shop, let

$$T_j = \textstyle\sum_{1 \leq i \leq n} a_{ij} = \text{total time needed on processor } j,\ 1 \leq j \leq m$$
$$L_i = \textstyle\sum_{1 \leq j \leq m} a_{ij} = \text{length of job } i,\ 1 \leq i \leq n$$

Hence any preemptive schedule must have a finish time of at least

$$\alpha = \max_{i,j}\{T_j, L_i\}. \tag{4.1}$$

The preemptive m-processor scheduling algorithm proposed in [26] always has a finish time of α. α corresponds to the lower bound on schedule length of (3.6).

The algorithm constructs a bipartite graph, it consists of two vertex sets $X = \{J_1, \ldots, J_{n+m}\}$ and $Y = \{M_1, \ldots, M_m\}$. The m jobs, $\{J_{n+1}, \ldots, J_{m+n}\}$, added to the job set, represent fictitious jobs and a set of edges connect J_{n+j} to M_j, $1 \leq j \leq m$ to make the sum of all incoming edges into each node M_j equal to α. The slots during which processors are assigned to these fictitious jobs represent processor *idle* times.

For every job i, its slack time is defined to be the difference between the amount of time remaining in the schedule and the amount of processing left for that job. A job is said to have become *critical* once its slack time goes to zero, at which time the job has to be included into the schedule for it to complete by the finish time.

The algorithm starts by computing an initial complete matching [25] of Y into X at time $= 0$. This matching is used as a starting point and may contain arbitrary job-processor assignment. The algorithm then identifies the time for which the current matching could be used as a schedule. Once a job not in the schedule becomes critical or one of the task in the matching finishes its processing, all non-critical jobs are removed from the matching, and augmenting paths [24] from all critical jobs are computed. The symmetric difference between the current matching and the augmenting paths forms the next matching. In case the matching computed is not complete, the jobs that were removed from the previous matching are reconsidered and augmenting paths from jobs not in the current and previous matching are computed to get a complete match. The process above is repeated until the processor requirements of all the jobs is met.

[26] shows that the algorithm always has a finish time of α and prove the correctness of the algorithm that a complete matching can be obtained during every iteration of the algorithm. The asymptotic time complexity of the algorithm is $O(r(\min\{r, m^2\} + m\log n))$, where n is the number of jobs, m the number of processors, and r the number of nonzero tasks.

The example in Table 4.1 considers a three-processor open shop problem with five jobs and the following task times:

Processor	Job					T
	1	2	3	4	5	
1	9	5	0	0	2	16
2	9	2	6	7	7	31
3	1	8	7	3	8	27
L	19	15	13	10	17	$\alpha = \max_{i,j}\{T_j, L_i\} = 31$

Table 4.1: Collapsed demand matrix for three processors and five jobs

Additions of the fictitious jobs J_6, J_7 and J_8, introduces three more columns into Table 4.1:

$$\begin{bmatrix} 15 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

As mentioned above, the weights assigned to the edges between the fictitious jobs and the processors represent idle slots. The entries in the additional columns suggest that processor 1 will have 15 idle slots and processor 3 will have 4 idle slots.

The Figure 4.1 shows the constructed bipartite graph for the three-processor open-shop. At each step of the algorithm a matching of size three is obtained representing a task

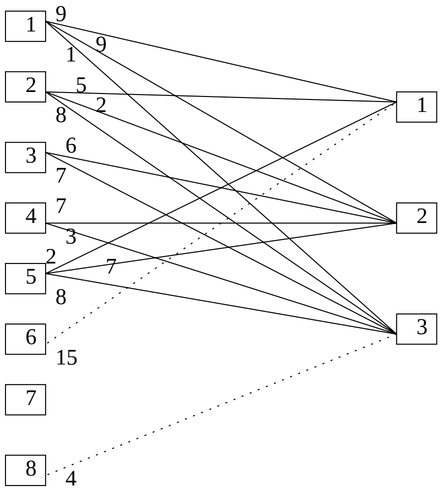

Figure 4.1: Bipartite graph as constructed by the preemptive open-shop algorithm

being assigned to each of the three processors. The Figure 4.2 shows the resulting open-shop schedule. The initial matching assigns jobs, J_6, J_5 and J_8 to processors M_1, M_2 and M_3 respectively. At time 4, job J_8 finishes its execution on processor M_3, hence it is deleted from the matching and an augmenting path starting from a job (J_4) not in the matching is computed. At time 7, job J_4 finishes its execution on processor M_3, and so does job J_5 on processor M_2, augmenting paths starting from jobs not in the matching assign jobs J_4, J_5 to processors M_2, M_3 respectively. At time 12, job J_1 becomes critical and hence replaces job J_5 on processor M_3. Later at time 13, job J_1 finishes its processing on M_3, but remains critical hence is assigned M_2 and job J_5 regains processor M_3. The next interesting instance is at time 21, at which job J_2 becomes critical and is assigned processor M_3. Next at time 23, job J_3 becomes critical and is assigned processor M_2. It should be noted that once a job becomes critical it is assigned a processor till the end of schedule.

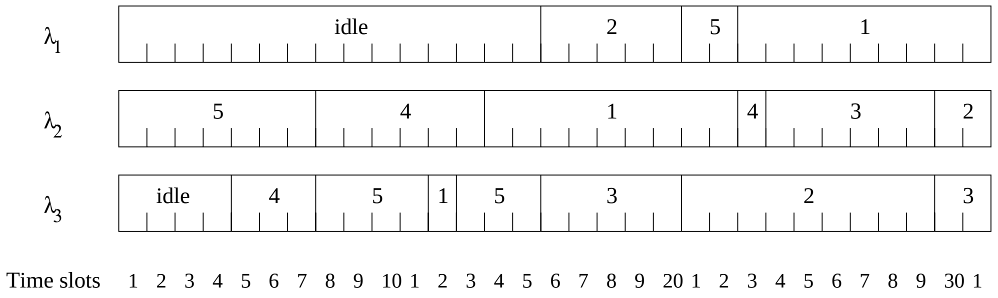

Figure 4.2: Preemptive open-shop schedule without best effort traffic, for $N = 5$ and $C = 3$

Best effort allocation for the preemptive open-shop algorithm (BE-OS)

Input: N, C, Collapsed traffic matrix $\mathbf{A}[N \times C]$, BE requirement matrix $\mathbf{BEReq}[N \times C]$, additional slots F

Output: Best effort allocation matrix $\mathbf{B}[N \times C]$

```
1.  begin
2.      $T_j = \sum_{1 \le i \le N} a_{i,j}$, $1 \le j \le C$
3.      $L_i = \sum_{1 \le j \le C} a_{i,j}$, $1 \le i \le N$
4.      $\alpha = \max_{i,j}\{T_j, L_i\}$.
5.      Set $SchLgt = \alpha + F$
6.      while $T_j < SchLgt$ and $L_i < SchLgt$ $\forall i, j$ $1 \le i \le N$, $1 \le j \le C$
7.          Start with a random node and channel, $ri, rj$
8.          for all nodes, $i$, and channels, $j$, starting from $ri, rj$
9.              if $\mathbf{BEReq}[i][j] == 1$ and $T_j < SchLgt$ and $L_i < SchLgt$ then
10.                 increment $b_{ij}, T_j, L_i$
11.             end if
12.         end for
13.     end while
14. end
```

Figure 4.3: Algorithm: Best-effort allocation for preemptive open-shop algorithm (BE-OS)

4.2 Best-effort traffic allocation for *OS*

As can be seen above only the channel(s) which have total demand equal to the OFT, α, or the node(s) whose sum of reservations on all the channels equals the OFT, α, are kept busy all the time. We propose an algorithm to distribute excess bandwidth on the non-busy channels to the nodes with best-effort traffic, in a *max-min fair* manner. The basic idea of the algorithm is to allocate the idle slots to the nodes with best-effort traffic such that neither the total time needed on channel, $T_j, 1 \le j \le m$ nor the total demand of a node, $L_i, 1 \le i \le n$, exceeds the optimal schedule length, α.

The Figure 4.3 contains the algorithm for best-effort allocation for the preemptive open-shop scheduler (BE-OS). The algorithm takes as input the number of nodes, N, number of channels, C, the collapsed traffic demand matrix, $\mathbf{A}$, the best effort requirement matrix, **BEReq**, and the number of additional slots to be allocated, F. If node i has best effort traffic destined to channel j, then $\mathbf{BEReq}[i][j]$ is set to 1 and 0 otherwise. The parameter F represents the number of slots, in addition to α, by which to increase the schedule length. In the case where F is zero only the idle slots will be assigned for best effort traffic. This parameter is required since many network providers set aside some percentage of bandwidth for best effort traffic (hence F could have been specified as a percentage of the total schedule length instead of number of slots). The algorithm starts every iteration of slot allocation at a random node-channel pair so that none of the node-channel pairs get a positional advantage.

In the case where all the nodes have best effort traffic on each of the channels, the preemptive open-shop alogorithm can be further optimized, in that, it does not require to add the ficititious jobs that represent idle slots. This change will not affect the asymtotic time complexity of the preemptive open-shop scheduling algorithm, however it will improve its running time.

Extending the example used in Section 4.1 (demand matrix shown in Table 4.1), for the best effort requirement matrix shown on the left below, the matrix on the right shows the best effort slots allocated by *BE-OS*.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & 5 & 0 & 5 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 1 \end{bmatrix}$$

The best effort allocation conforms to the definition of *max-min fair* allocation in *multiple* resource situations [33]. The combined matrix, $\mathbf{T}$, is obtained by the addition of the collapsed traffic matrix and the best effort allocation matrix (Equation (3.2)). The three-processor open shop problem becomes as shown in Table 4.2.

It can be noticed that the total time needed on each channel (processor) is equal to the schedule length, α. The resulting schedule as computed by the preemptive open-shop scheduling algorithm is shown in the Figure 4.4, as can be noticed it has no idle time slots.

Neither the preemptive open-shop scheduling algorithm nor the best effort allocation algorithm assume any sort of traffic distribution. In general, the best effort slot alloca-

Channel	Node					T
	1	2	3	4	5	
1	9	10	0	5	7	31
2	9	2	6	7	7	31
3	1	8	9	4	9	31
L	19	20	15	16	23	$\alpha = \max_{i,j}\{T_j, L_i\} = 31$

Table 4.2: Combined traffic matrix, with best effort slots allocated by BE-OS

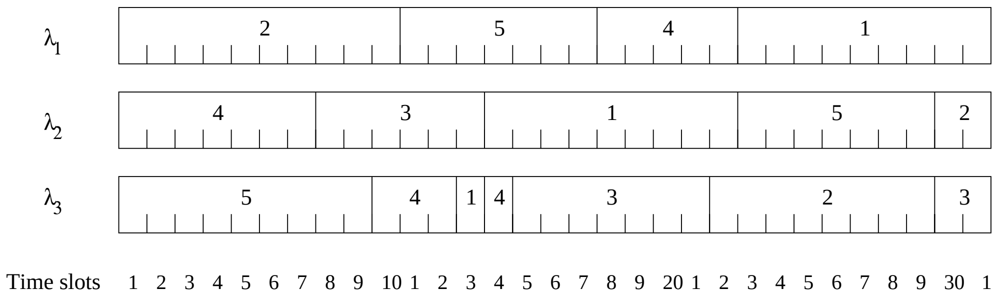

Figure 4.4: Preemptive open-shop schedule with best effort traffic, for $N = 5$ and $C = 3$

tion scheme proposed increases the channel throughput, $\mathcal{S} \approx C$. However the throughput may be lower than C when the best effort requirement matrix is sparse or best effort requirements are required on busy channels or by busy nodes. Simulation results (see Section 4.5) show that schedules with best-effort allocation obtain a channel throughput of almost C even for low probabilities of best-effort requirement.

4.3 Non-preemptive Open-Shop with tuning latencies

[28] considers the problem of scheduling packets transmissions in a broadcast, single-hop WDM network with arbitrary transmitter tuning latencies. [28] proposes a *non-preemptive* schedule, since under such a schedule, each transmitter will tune to each channel exactly once, minimizing the overall time spent in tuning. [28] refers to the problem of determining an optimum length schedule for the collapsed traffic matrix $\mathbf{A}$, with tuning slots $\Delta \geq 0$ as the *Open-Shop Scheduling with Tuning Latencies (OSTL)* problem. *OSTL* is a generalization of the nonpreemptive open-shop scheduling (*OS*) problem [26]; it reduces to the later when we let $\Delta = 0$. Problem *OS* is $\mathcal{NP}$-complete when the number of wavelengths

$C \geq 3$ [26]. But for $C = 2$, there exists a polynomial-time solution of *OS*. [28] proves that OSTL is $\mathcal{NP}$-complete for any $C \geq 2$, hence proving that OSTL is more difficult than OS.

We consider the system to be operating in the *bandwidth-limited* region (see Chapter 3). In a bandwidth-limited network the length of the schedule, satisfying the *no-collision* and *transmitter* constraints, cannot be smaller than the number of slots required to satisfy all transmissions on any given channel, hence the overall lower bound on the length of schedule is given by:

$$M^{(l)} = M_{bw}^{(l)} = \max_{1 \leq c \leq C} \left\{ \sum_{i=1}^{N} a_{i,c} \right\} \tag{4.2}$$

[28] calls $s_1 = (\pi_1, \pi_2, \ldots, \pi_N)$ as a *transmitter sequence* for a channel λ_1 if π_2 is the first node after π_1 to transmit on λ_1, π_3 is the second such node, and so on. And the sequence repeats, that is, node π_1 transmits on λ_1 after node π_N. Similarly, $v_1 = (\lambda_{\pi_1}, \lambda_{\pi_2}, \ldots, \lambda_{\pi_C})$ is said to be a *channel sequence* for node 1, if this is the order in which node 1 is assigned to transmit on the various channels, starting with channel λ_{π_1}. Given S a schedule of length M for the collapsed traffic matrix $\mathbf{A}$, the transmitter sequences for each channel are completely specified. In general, these sequences can be different for the various channels. However, [28] considers a class of schedules in which the transmitter sequences are same for all the channels. (Equivalently the class of schedules such that the channel sequences are same for all the transmitters.)

[28] formulates the problem of finding a optimum schedule for a given transmitter sequence as an *integer programming* problem. The key idea of the *scheduling* algorithm, which the authors call *Make_Bandwidth_Limited_Schedule (MBLS)*, is to schedule transmissions on λ_1 so that this channel is always kept busy, except perhaps after all the nodes have been given a chance to transmit. The algorithm first determines the channel sequence by ordering the channels in the decreasing order of T_j, $1 \leq j \leq C$. It then determines the transmitter sequence by ordering the transmitters in the decreasing order of L_i, $1 \leq i \leq N$. Hence we now have the channel λ_1 as the dominant channel, that is, $\sum_{i=1}^{N} a_{i,1} = M^{(l)}$. The algorithm executes in two passes. All gaps[1] in the dominant channel λ_1 are initialized to zero; then in Pass 1, transmissions in channels λ_2 through λ_C are scheduled at the earliest possible time that satisfies the *no-collision* and *transmitter* constraints. Doing so, however, may introduce large gaps into these channels, resulting in a sub-optimal schedule. During

[1]Gaps are defined as the number of slots that a channel remains idle between the end of transmissions by node i and start of transmissions by node $i + 1$.

the second pass, the algorithm attempts to compact the gaps within each channel by shifting the slots to the right or left, but only so far as the *no-collision* and *transmitter* constraints allow. The time complexity of the MBLS algorithm is $O(C.N^2)$.

Consider the example used in Section 4.1, with the collapsed traffic matrix in Table 4.1, shown along with the values for T_j and L_i. For a system with five nodes, three wavelengths and transmitter tuning latency of $\Delta = 2$ slots, the schedule as determined by the MBLS algorithm is shown in Figure 4.5.

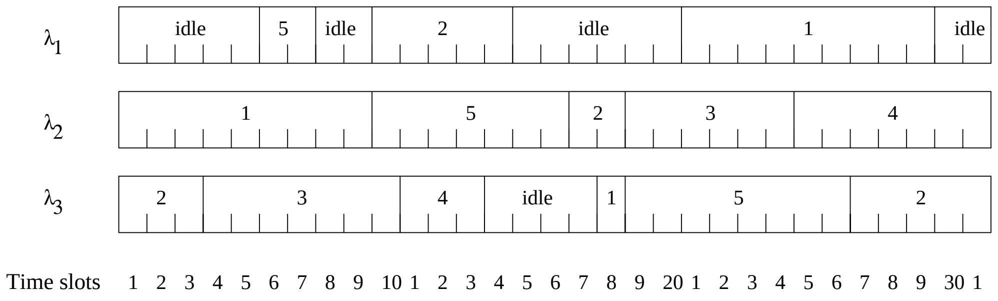

Figure 4.5: Nonpreemptive OSTL schedule without best effort traffic, for $N = 5$, $C = 3$ and $\Delta = 2$

The algorithm starts by ordering the channels in the order of T_j, $1 \leq j \leq 3$. The channel sequence thus formed is $\{\lambda_2, \lambda_3, \lambda_1\}$. The algorithm then orders the transmitter in the order of L_i, $1 \leq i \leq 5$, the transmitter sequence thus formed is $\{1, 5, 2, 3, 4\}$. Transmitters in the transmitter sequence are then assigned to channel λ_2 such that the channel is always kept busy. Transmitters in the transmitter sequence are then assigned slots on the channel λ_3 such that the transmitters have at least two slots between the end of transmissions on channel λ_2 and start of transmissions on λ_3. This operation introduces a gap of six slots between the end of transmission by node 1 and the start of transmission by node 5. Similarly transmitters are then assigned slots on the last channel, λ_1. Again this operation introduces a gap of five slots between the end of transmission by node 1 and start of transmission by node 5, and a gap of six slots between the end of transmission by node 5 and start of transmission by node 2. The second Pass of the algorithm pushes the slot assigned to transmitter 1 on channel λ_3 six slots to the right, thus making space for the 13 slots of the schedule to wrap around. Similarly it moves the slots assigned to node 1 on channel λ_1 six slots to the right to allow node 1 to tune in from channel λ_3. The resulting schedule is shown in Figure 4.5.

It should be noticed that the algorithm MBLS was able to schedule the collapsed traffic matrix of Table 4.1 to obtain the optimal schedule length. However, it has quite a few idle slots due to the varying demands on the three channels. We now explore how these slots could be assigned to best-effort traffic sources without increasing the schedule length.

4.4 Best-effort traffic allocation for *OSTL*

[28] provides an upper bound on the "degree of nonuniformity" of the collapsed traffic matrix, **A**, in order to guarantee a schedule of length equal to the lower bound. [28] proves that for a collapsed traffic demand matrix, **A**, in a bandwidth-limited network, a schedule of length equal to $M^{(l)}$ exists within the class of schedules for a given transmitter sequence, if the elements of **A** satisfy the following condition:

$$\left| a_{ic} - \frac{M^{(l)}}{N} \right| \leq \epsilon \, \forall i, c \tag{4.3}$$

with ϵ given by

$$\epsilon = \frac{2M^{(l)}}{N+2} \left(\frac{1}{C} - \frac{1}{N} - \frac{\Delta}{M^{(l)}} \right). \tag{4.4}$$

It can be noticed that to have a higher value of ϵ the network needs to be designed such that the lower bound on schedule, $M^{(l)}$ is high, that is, higher a_{ic} values. Also smaller the number of channels, C, higher would be the value of ϵ. Also a higher value ϵ could be obtained by having the number of nodes, N, significantly higher than the number of channels, C. For instance, for $N = 200, C = 24$, and ignoring the $\frac{\Delta}{M^{(l)}}$ term, we get $\frac{\epsilon}{M^{(l)}/N} \approx .062$. That is, the elements a_{ic} could vary 6.2% around $\frac{M^{(l)}}{N}$ and the algorithm would still guarantee a schedule of length $M^{(l)}$.

However, since we do not assume a uniform distribution for the collapsed matrix, **A**, elements could be higher than the average slot reservation by a number greater than the "degree of nonuniformity". Hence in the best-effort allocation algorithm that we propose, we only consider elements, a_{ic}, that have demand strictly less than the *average slot requirement* $(\frac{M^{(l)}}{N})$. The basic idea of the algorithm is to allocate the excess bandwidth (idle slots) to nodes which have a lower than average slot requirement. Figure 4.8 shows the best effort alloction algorithm for the nonpreemptive OSTL algorithm (BE-OSTL).

The algorithm takes as input the number of nodes, N, the number of channels, C, the collapsed traffic demand matrix, **A** and the number of additional slots, F, to be allocated. The algorithm identifies the elements that have less that the average slot requirement, and tries to allocate additional slots to such elements without increasing the total transmission time required on the channel, beyond the lower bound on the length of schedule, $M^{(l)}$. The term $\frac{Alloc_{i,j}}{pos\_eT_j}$ on line 14 of the algorithm signifies the percentage of excess slots on the channel λ_j that can be assigned to node i. The time complexity of the algorithm is $O(N.C)$.

Again consider the three channel, five node example with the collapsed traffic matrix in Table 4.1. The BE-OSTL algorithm first computes the excess slots that a node can get on each of the channels, it then computes the total number of excess slots on each channel. The table below shows excess time slots that a node could be allocated on each of the channels, a negative value for the element suggesting that the corresponding element in the collapsed traffic matrix, **A**, exceeds the average slot requirement of 6 ($\frac{M^{(l)}}{N}$). The eT column contains the sum of all the elements of a row in the excess slot allocation matrix, it adds both the positive and negative values. Similarly, the eL row contains the sum of all the elements of a column in the excess slot allocation matrix. The pos_eT column and the pos_eL row contain sums of only the positive elements of the excess slot allocation matrix.

Channel	Node					eT	pos_eT
	1	2	3	4	5		
1	-3	1	6	6	4	14	17
2	-3	4	0	-1	-1	-1	4
3	5	-2	-1	3	-2	3	8
eL	-1	3	5	8	1		
pos_eL	5	5	6	9	4		

The best effort slot allocation for a node i on channel λ_j are computed using the element $Alloc_{i,j}$ such that it does not increase the total transmission time on a channel λ_j beyond the lower bound. For instance, node 3 on channel λ_1 is allocated $\min(14 * \frac{6}{17}, 5 * \frac{6}{6}) = \min(\lfloor 4.9 \rfloor, 5) = 4$. The best effort allocation is shown in the matrix below:

$$\begin{bmatrix} 0 & 0 & 4 & 4 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The best effort traffic allocation, matrix **B**, is then added to the collapsed traffic matrix, **A** to yield the combined traffic matrix, **T**. The nonpreemptive OSTL algorithm operates on this combined traffic matrix. The resulting schedule is shown in figure 4.6.

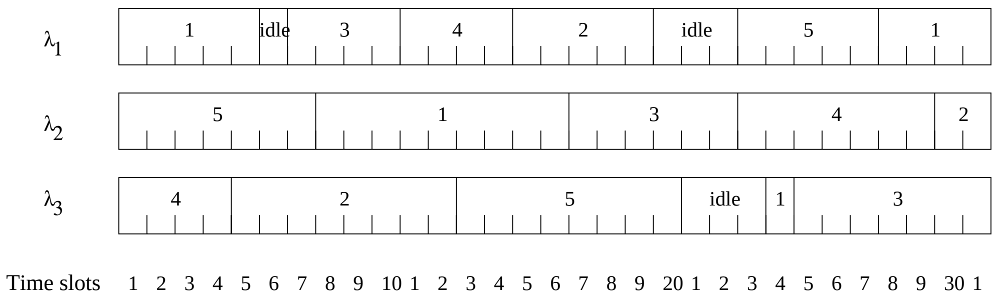
λ1
1
idle
3
4
2
idle
5
1
λ2
5
1
3
4
2
λ3
4
2
5
idle
1
3
Time slots 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1

Figure 4.6: Nonpreemptive OSTL schedule with best effort traffic, for $N = 5$, $C = 3$ and $\Delta = 2$

It can be noticed that the schedule still has some idle slots, and it would be tempting to rather use the *BE-OS* best effort algorithm proposed in Section 4.2. However, the *BE-OS* algorithm does not discriminate the best effort slot allocation depending on the individual elements of the collapsed traffic matrix, $\mathbf{A}$. That is, node i that has demand a_{ic} on channel c greater than the average slot requirement is also considered for best effort slot allocation. Doing this may result in the resulting nonpreemtive OSTL schedule length being greater than the lower bound, $M^{(l)}$. The figure 4.7 shows the nonpreemptive OSTL schedule for the combined traffic matrix of Table 4.2, which was obtained by the addition of best effort slots allocated by the *BE-OS* algorithm, to the collapsed traffic matrix in Table 4.1. The figure shows that the length of the schedule is 35 slots, which is greater than the lower bound of 31 slots. Simulation results (see Section 4.5) further corroborate that *BE-OSTL* scheme is better suited for best effort allocation when the nonpreemptive *MBLS* scheduling algorithm is used to schedule packets.

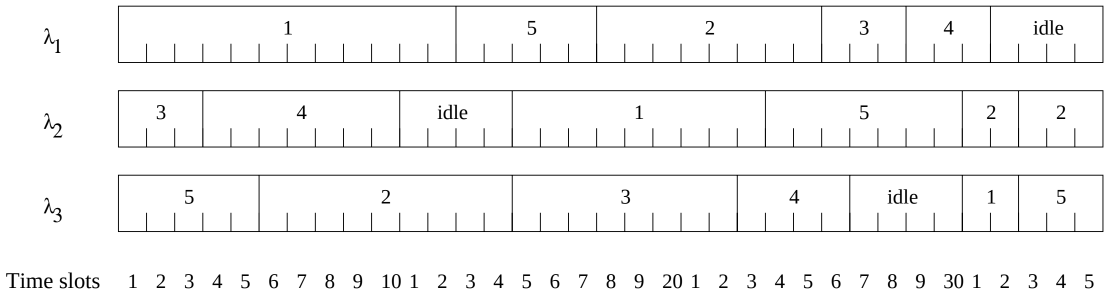
λ1
1
5
2
3
4
idle
λ2
3
4
idle
1
5
2
2
λ3
5
2
3
4
idle
1
5
Time slots 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 20 1 2 3 4 5 6 7 8 9 30 1 2 3 4 5

Figure 4.7: Nonpreemptive OSTL schedule with best effort traffic allocation by BE-OS, for $N = 5$, $C = 3$ and $\Delta = 2$

Algorithm *BE-OS* has the advantage that it can be used to allocate best effort slots to nodes that specifically request best-effort slots (via the best effort requirement matrix, **BEReq**. Further it places no restriction on which nodes get the excess slots, contingent that the total transmission on the channel or the total demand of a node does not exceed the optimal schedule length. However, algorithm *BE-OS* is designed to exploit the characteristics of preemptive scheduling algorithms, which can be employed in systems with negligible tuning latency as compared to the packet transmission time ($\delta \ll 1$). Algorithm *BE-OSTL* provides an efficient mechanism of allocating excess slots in a system employing nonpreemptive scheduling algorithms.

4.5 Simulation Study

In this section, we study the performance of the two best effort allocation schemes, *BE-OS* and *BE-OSTL*, presented in Sections 4.2 and 4.4 of this chapter. We also compare the performance of the two best effort allocation schemes when the nonpreemptive *MBLS* algorithm is used to schedule packets.

We generate random instances of the traffic demand matrix, **A**. The elements of the matrix **A** are generated by an exponential random number generator with mean of 12 arrivals, and a maximum value of 21. Each point plotted represents the average of results obtained by applying the algorithms on six thousand independent matrices **A**. The averages are plotted along with 95% confidence interval values[2]. We have plotted graphs for a number of channels $C = 6, 24$, and a number of nodes N, between 9 and 20 for $C = 6$, and between 60 and 150 for $C = 24$. Other associated variables will be specified during the discussion of the corressponding graphs. The channel throughput, $\mathcal{S}$, is defined as the average number of packet transmissions on the C channels in a time slot. $\mathcal{S}$ can be obtained as the ratio of total number of non-idle slots in a schedule to the schedule length. We define channel utilization as:

$$\frac{\mathcal{S}}{C} \times 100 \tag{4.5}$$

Channel utilization represents how much percentage of the total available bandwidth is assigned for packet transmissions. Figures 4.9 and 4.10 plot the channel utilization (4.5) with *BE-OS* best effort allocation when the preemptive open-shop scheduling algorithm is

[2]The confidence interval values were obtained by the method of replications, each individual run is independent of other runs. The confidence interval is: $(\bar{x} - 1.96\frac{s}{\sqrt{n}},\ \bar{x} + 1.96\frac{s}{\sqrt{n}})$ where $\bar{x}$ and s are the sample mean and variance of n values, respectively.

used to schedule packets. Figures 4.11 and 4.13 plot the channel utilization (4.5) with *BE-OSTL* best effort allocation when the nonpreemptive *MBLS* algorithm [28] is used to schedule packets.

The lower bound on the schedule length of a bandwidth-limited network, $M^{(l)}$, can be obtained from (4.1). Let M be the actual length of schedule, for a given traffic matrix, produced by some scheduling algorithm. Then the quantity

$$\frac{M - M^{(l)}}{M^{(l)}} \times 100 \tag{4.6}$$

represents how far the length M of the schedule produced by the algorithm is from the lower bound. Figure 4.15 and Figure 4.16 plot (4.5) and (4.6), respectively, against the number of nodes, N, for channel throughput and schedule length comparisions between the *BE-OS* and *BE-OSTL* best effort allocation algorithms when the nonpreemptive *MBLS* algorithm is used to schedule packets.

Let $M_{\widetilde{BE}}$ be the actual length of a schedule, for a given traffic matrix without best effort allocation, produced by some scheduling algorithm, say A. Let M_{BE} be the actual length of a schedule, for the given traffic matrix combined with the best effort allocations, produced by the same scheduling algorithm, A. Then the quantity

$$\frac{M_{BE} - M_{\widetilde{BE}}}{M_{\widetilde{BE}}} \times 100 \tag{4.7}$$

represents the percentage increase in the schedule length after combining a given traffic matrix with best effort allocation, the schedules being produced by the same scheduling algorithm. Figures 4.12 and 4.14 plot the quantity in (4.7) against the number of nodes, N, *BE-OSTL* being the best effort allocation scheme and the schedule being computed by the nonpreeemptive *MBLS* algorithm.

Performance of *BE-OS* with preemptive open-shop scheduling

In addition to the traffic demand matrix, $\mathbf{A}$, the *BE-OS* algorithm takes as input a best effort requirement matrix, **BEReq**, which conveys which nodes need best effort allocations and on which channels. Each element of the matrix **BEReq** is set to 1 with a uniform probability of p_{be}, that is, node i has best effort traffic on channel j, $1 \leq i \leq N, 1 \leq j \leq C$, with probability p_{be}. The Figures 4.9 and 4.10 plot channel utilization (4.5) against the number of nodes, N, for values of $p_{be} = \{0, 0.25, 0.5, 0.75\}$[3]. The case with $p_{be} = 0$ represents the

[3]The case $p_{be} = 1$ is always guaranteed to yield 100% channel utilization.

schedules without best effort allocation. As expected, such schedules yield lower channel utilization irrespective of the value of N and C. The lower values of p_{be} represent sparse best effort requirements, and it is possible for these requirements to be on busy channels or by busy nodes. Hence the resulting schedules may have idle slots, yielding a lower that 100% channel utilization. However as the number of nodes increases the simulation results show that even for low values of p_{be} all the available bandwidth could be utilized for packet transmissions, that is, channel throughput $\mathcal{S}$ approaches C.

As discussed in Section 4.2, the best effort allocation by algorithm *BE-OS* does not increase the schedule length of the resulting combined traffic matrix, that is, the quantity in (4.7), for *BE-OS* best effort allocation with the preemptive open-shop scheduling algorithm, is always zero.

Best effort allocation for nonpreemptive *MBLS* scheduling

Figures 4.11 and 4.13 plot the channel utilization (4.5) obtained by the nonpreemptive *MBLS* scheduling algorithm for values of tuning latency $\Delta = 0, 4$, with and without best effort allocations. The *BE-OSTL* best effort allocation scheme is used to allocate best effort slots to nodes that have lower than the average slot requirement of $\frac{M^{(l)}}{N}$. The figures show that the best effort allocation by the *BE-OSTL* scheme increases the channel utilization in both the cases with tuning latency $\Delta = 0$ and $\Delta = 4$. It is equally important that the best effort allocation of slots should not increase the schedule length considerably. Figures 4.12 and 4.14 plot the quantity in (4.7), which represents the percentage increase in schedule lengths after best effort allocation. As can be seen the *BE-OSTL* scheme does not increase the schedule lengths considerably. It can be observed that as the number of nodes increases the channel utilization increases and the percentage increase in the schedule length decreases.

We now consider the comparisons between the *BE-OS* and *BE-OSTL* best effort allocation schemes, when the nonpreemptive *MBLS* scheduling algorithm is used to schedule packets. The *BE-OSTL* algorithm considers nodes for best effort allocation regardless of the best effort requirement of the nodes. Hence all the elements of matrix **BEReq** are set to 1, so that even the *BE-OS* scheme considers all the nodes for best effort allocation. Figure 4.15 plots the channel utilization obtained by two schemes for values to tuning latency $\Delta = 0, 4$. And Figure 4.16 plots the quantity in (4.6) which represents how far the schedule lengths are from the lower bound. The figures show that best effort allocations by the *BE-OS* scheme yield a slightly higher channel utilization, however it also results in an increased schedule length. Thus validating the need for the two different best effort allocation schemes. The *BE-OS* scheme is appropriate with preemptive *OS* packet scheduling algorithm while the *BE-OSTL* scheme operates well with the nonpreemptive *MBLS* packet scheduling algorithm.

In the next chapter we discuss optimization heuristics that yield better *delay* and *delay jitter* performance in systems which employ preemptive packet scheduling mechanisms.

Best effort allocation for the nonpreemptive OSTL algorithm (BE-OSTL)

Input: N, C, Collapsed traffic matrix $\mathbf{A}[N \times C]$, additional slots F

Output: Best effort allocation matrix $\mathbf{B}[N \times C]$

```
1.  begin
2.      $T_j = \sum_{1 \le i \le N} a_{i,j}$,          $1 \le j \le C$
3.      $L_i = \sum_{1 \le j \le C} a_{i,j}$,          $1 \le i \le N$
4.      $\alpha = \max_{i,j}\{T_j, L_i\}$.
5.      Set $SchLgt = \alpha + F$
6.      Set $UniAlloc = \frac{SchLgt}{N}$
7.      Set $Alloc_{i,j} = UniAlloc - a_{ij}$, $\forall i, j$, $1 \le i \le N$, $1 \le j \le C$
        // let $eT_j$ and $eL_i$ hold the excess bandwidth on each channel and node, respectively
8.      $eT_j = \sum_{i=1}^{N} Alloc_{i,j}$        $1 \le j \le C$
9.      $eL_i = \sum_{j=1}^{C} Alloc_{i,j}$        $1 \le i \le N$
        // elements with positive $Alloc_{i,j}$ are eligible for best-effort allocation.
        // let $pos\_eT_j$ and $pos\_eL_i$ hold the sum of possible allocation of all eligible channels
        // and nodes, in general, $eT_j \le pos\_eT_j$ and $eL_i \le pos\_eL_i$ $\forall i, j$
10.     $pos\_eT_j = \sum_{i=1}^{N}(Alloc_{i,j} > 0 ? Alloc_{i,j} : 0)$        $1 \le j \le C$
11.     $pos\_eL_i = \sum_{j=1}^{C}(Alloc_{i,j} > 0 ? Alloc_{i,j} : 0)$        $1 \le i \le N$
        // now allocate additional slots
12.     $\forall i, j$        $1 \le i \le N$, $1 \le j \le C$
13.         if $Alloc_{i,j} > 0$ and $eT_j > 0$ and $eL_i > 0$
14.             $b_{ij} = \min(eT_j * \frac{Alloc_{i,j}}{pos\_eT_j}, eL_i * \frac{Alloc_{i,j}}{pos\_eL_i})$
15.         end if
16.     end $\forall$
17. end
```

Figure 4.8: Algorithm: Best effort allocation for the nonpreemptive OSTL algorithm (BE-OSTL)

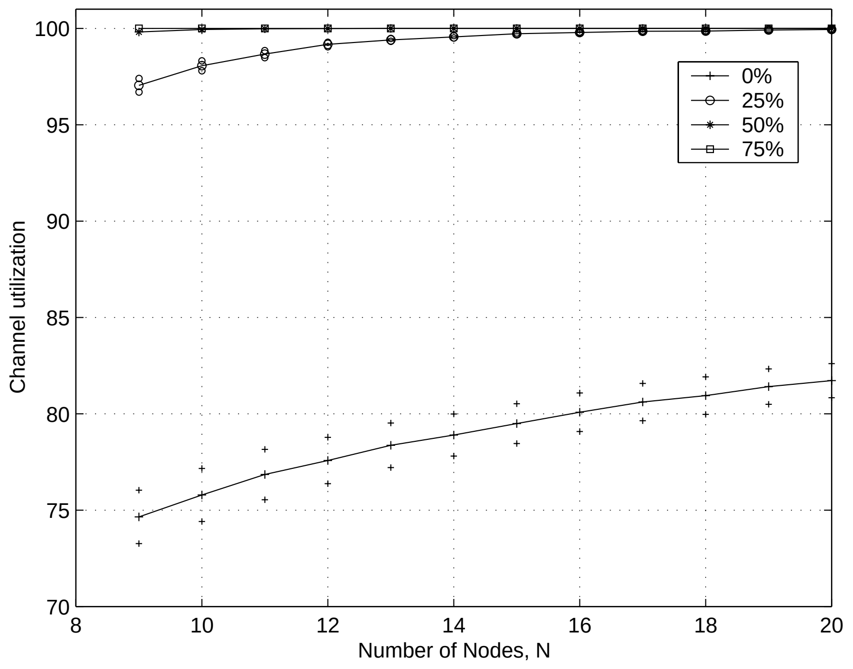

Figure 4.9: Channel throughput with BEOS best effort allocation, for C=6 (values plotted with 95% confidence interval)

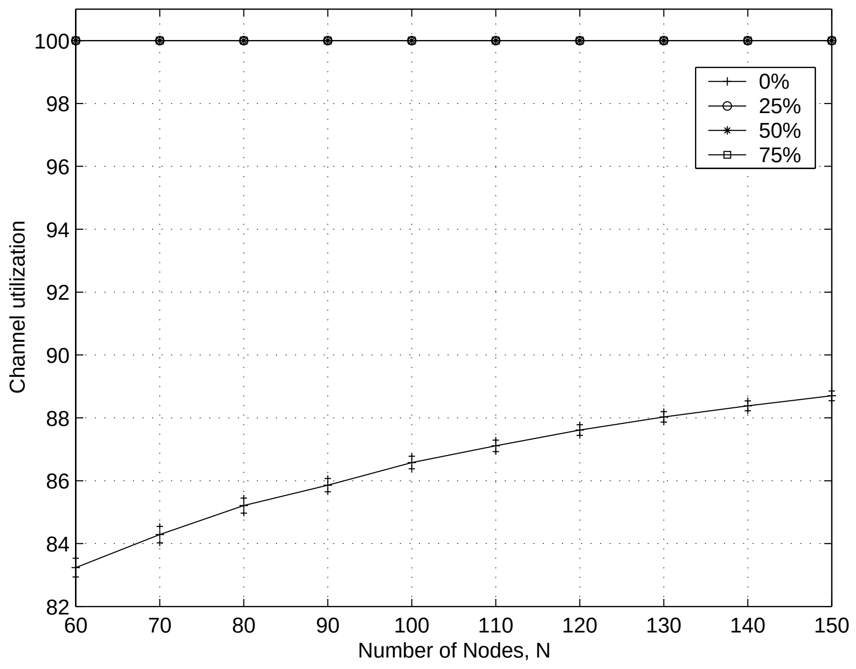

Figure 4.10: Channel throughput with BEOS best effort allocation, for C=24 (values plotted with 95% confidence interval)

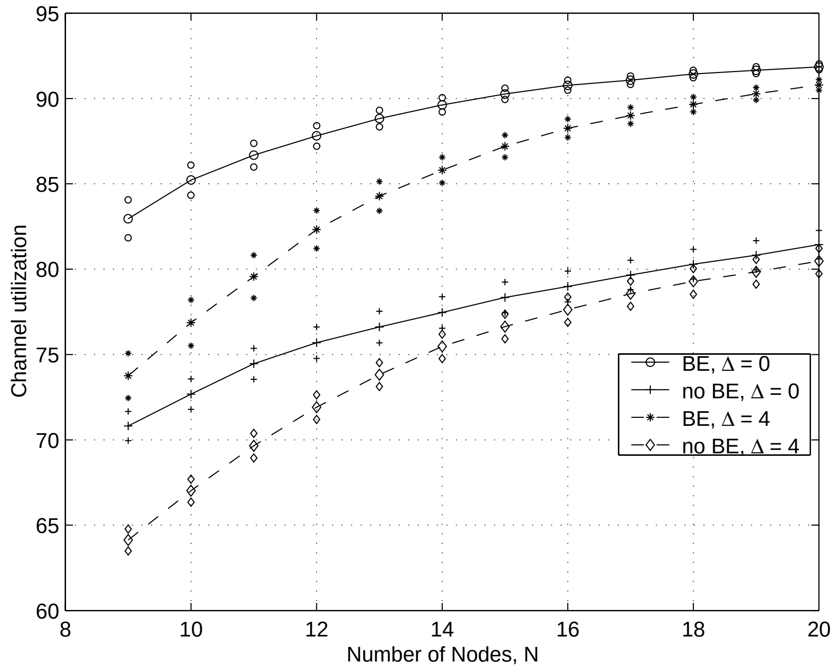

Figure 4.11: Channel throughput with BEOSTL best effort allocation, for C=6 (values plotted with 95% confidence interval)

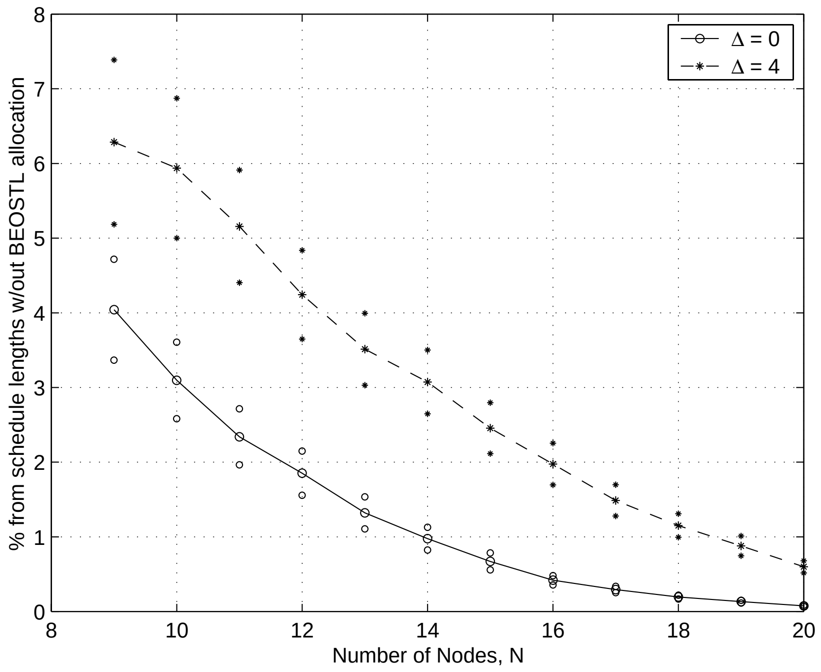

Figure 4.12: Percentage increase in schedule length after BEOSTL best effort allocation, for C=6 (values plotted with 95% confidence interval)

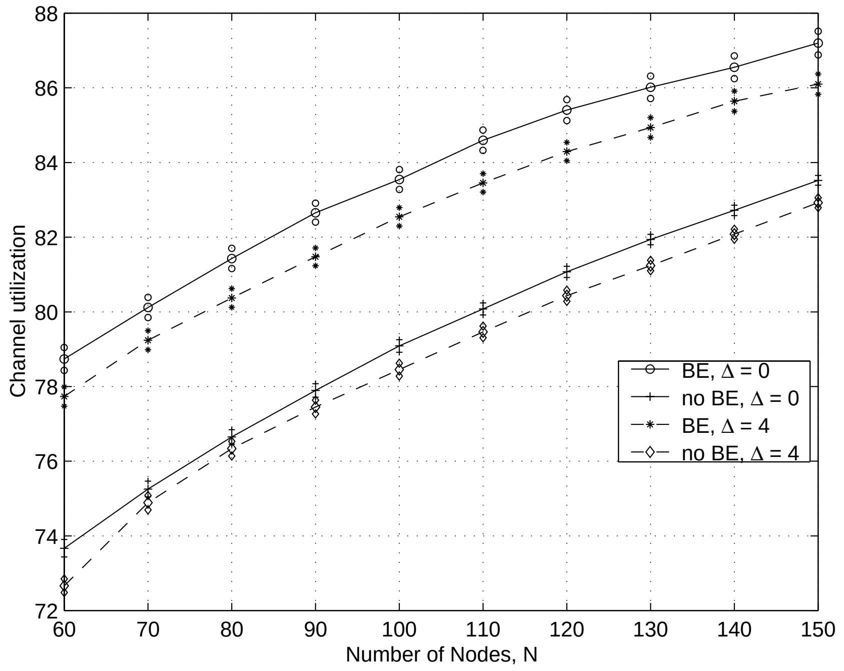

Figure 4.13: Channel throughput with BEOSTL best effort allocation, for C=24 (values plotted with 95% confidence interval)

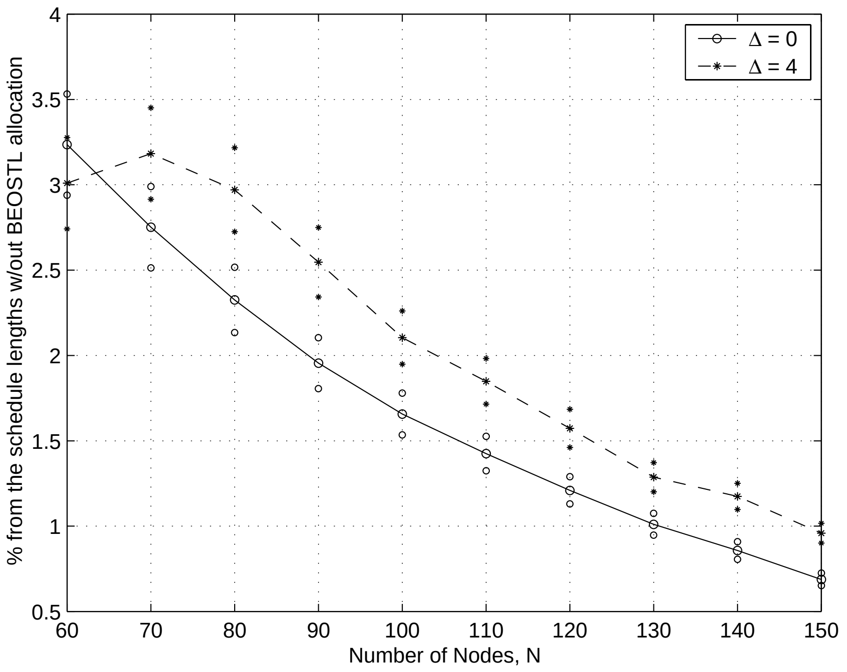

Figure 4.14: Percentage increase in schedule length after BEOSTL best effort allocation, for C=24 (values plotted with 95% confidence interval)

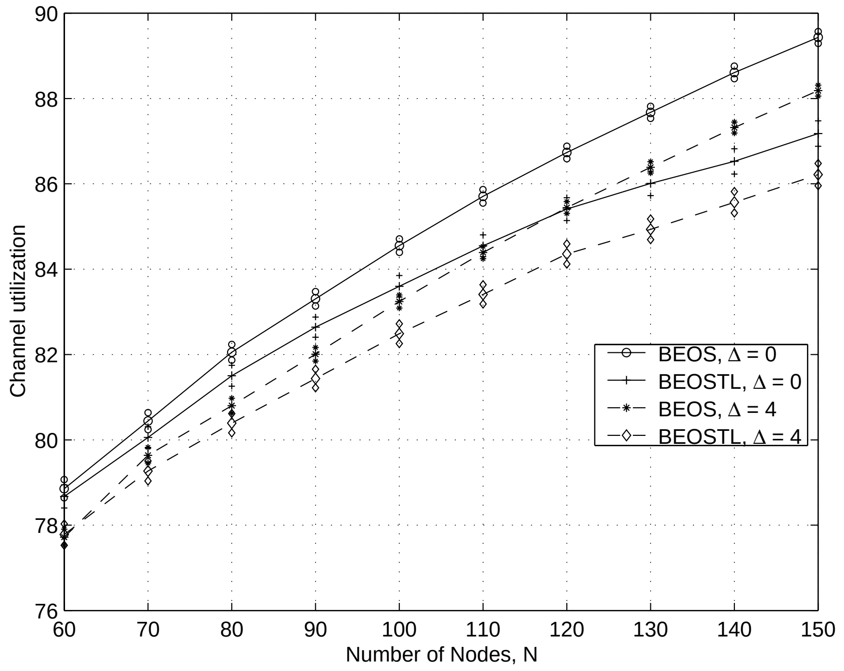

Figure 4.15: Channel throughput comparisions with OSTL scheduling algorithm, for C=24 (values plotted with 95% confidence interval)

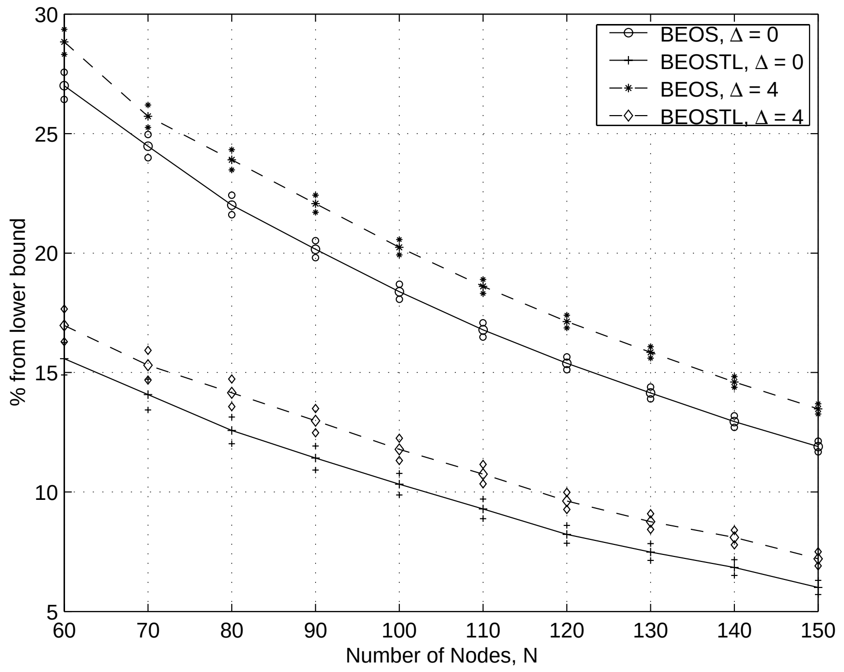

Figure 4.16: Schedule length comparisions with OSTL scheduling algorithm, for C=24 (values plotted with 95% confidence interval)

Chapter 5

Optimization Heuristics

The scheduling algorithms discussed in Chapter 4, schedule packets in bursts thus increasing the variance in delay experienced by the packet streams. Each node transmits a burst of packets in consecutive time slots during a cycle and then waits until its turn in the next schedule cycle. In Figure 5.1 the inter-departure time between packets transmitted at t_1 and t_2 is equal to the slot time (packet transmission time plus guard band, if any) and the inter-departure time between packets transmitted at t_2 and t_3 is equal to a schedule cycle time (schedule length, M, times the slot time). In this chapter we discuss a scheme that can reduce such variation in delay by scrambling the slots of a schedule so that adjacent slots are well separated. Thus decoupling the effect of schedule lengths on scheduling delay and delay-jitter.

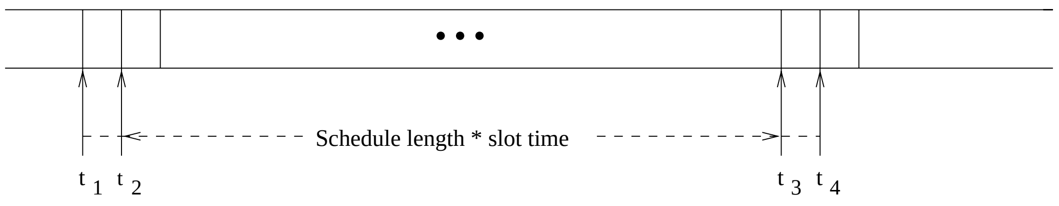

Figure 5.1: Effect of scheduling bursts of packets on scheduling delay

It makes sense to schedule packets in a burst if the objective of the scheduling algorithm is to minimize the time needed to satisfy the traffic demands and tuning time is not included as part of the slot time (for instance if the normalized tuning latency, $\delta \geq 1$ or $\delta \approx 1$). As described in Chapter 4, nonpreemptive scheduling algorithms aim to minimize the time transmitters spend in tuning, by tuning each transmitter to each channel exactly once. For such algorithms packets are transmitted in bursts, that is, such algorithms trade off the variance of delay experienced by the packets with reduction in tuning time.

However, systems which assume negligible tuning latency ($\delta \ll 1$), employ the preemptive scheduling algorithms. The algorithms may be so designed that they try to schedule packets in chunks of largest possible size. For instance, the optimal preemptive scheduling algorithm presented in Chapter 4 schedules a given matching (a set of node-to-channel assignment) until one of the node finishes its transmission or some other node becomes critical. Hence even the systems that employ preemptive packet scheduling algorithms may also be subject to the effects of delay variation. One could try to solve this problem by modifying the scheduling algorithm such that it schedules packets from different nodes in consecutive time slots. The other approach could be to scramble the slots of a schedule so that adjacent slots are well separated. We discuss such a mechanism in this chapter.

5.1 Channel Decomposition

Channel decomposition [36] is a technique that scrambles the time slots of a schedule such that adjacent slots are well spaced. The idea is to group slots in the odd (even) position together and then recursively apply the same rule on the newly formed group of slots until it no longer affects position of the slots (when group size equals one or two). The technique is demonstrated in Figure 5.2. As shown in the figure the channel decomposition is a recursive procedure, and the asymptotic time complexity of the procedure is $O(M \log_2 M)$, where M is the length of the schedule.

The scheduling algorithm computes the schedule as usual, once the schedule length is known, the channel decomposition procedure maps the adjacent slots to slots that are placed well apart. For instance consider the schedule of length 12 shown in Figure 5.3. Figure 5.2 shows the channel decomposition for the schedule of length 12. The decomposed channel suggests that slot 2 of the normal schedule maps to slot 9 of the decomposed schedule, similarly slot 3 of the normal schedule maps to slot 5, and so on. The second time line in Figure 5.3 shows the resulting decomposed schedule.

Simulation results show that channel decomposition effectively reduces the variation in the delay experienced by the packets. The effects are more pronounced for longer schedule lengths. Thus decoupling the effect of schedule lengths on scheduling delay and

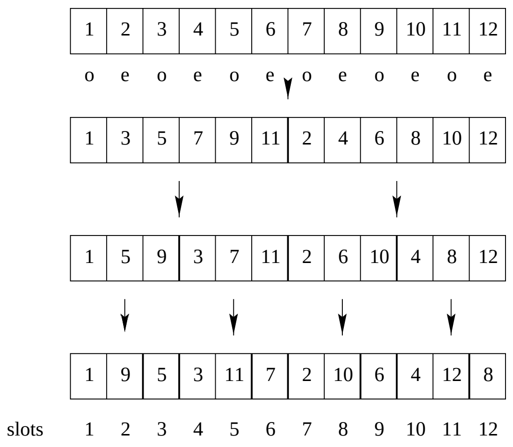

Figure 5.2: Channel Decomposition

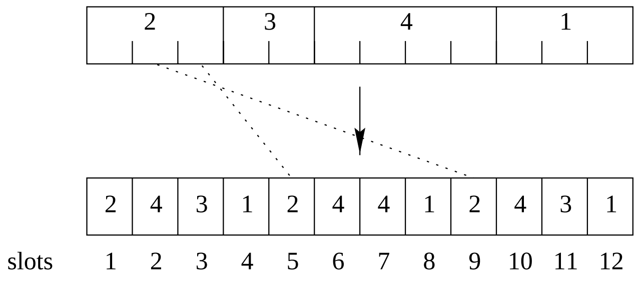

Figure 5.3: Decomposition of a 12-slot schedule

delay-jitter.

The next chapter discusses the architecture and implementation of the WDM single-hop *TT-FR* simulator component.

Chapter 6

Simulator Implementation

A significant contribution of this thesis is the implementation of a flexible and highly extensible simulator for tunable transmitter, fixed receiver WDM single-hop networks. The simulator component uses the functionality provided by the Diffserv model contributed by Nortel Networks [7] on *ns-2* [6]. This chapter explains the important aspects of our simulator. We start with a brief introduction to *ns-2* and the Nortel's Diffserv implementation. We then deal with the design and implementation of the WDM scheduler.

6.1 Introduction to *ns-2*

ns [6] has been developed at the Lawrence Berkeley National Laboratory (LBNL) of the University of California, Berkeley (UCB). Currently *ns* development and support is carried on at the Information Sciences Institute (ISI) of the University of Southern California, Los Angeles (USC). *ns* is a discrete event simulator targeted at networking research. *ns* provides substantial support for simulation of TCP and its variants, routing, and multicast protocols over wired and wireless (local and satellite) networks.

ns has an extensible background engine implemented in C++ that uses OTcl (an object oriented version of Tcl) as the command and configuration interface. *ns* uses the split programming paradigm because the simulator is designed to do two different kind of things. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. C++ meets the first requirement and OTcl the second.

Why integrate the WDM simulator component with *ns*?

ns-2 does not have intrinsic support for WDM links, i. e., multi-channel links. Hence integrating a WDM scheduler into *ns*, requires either adding the multi-channel functionality to the links or mapping the existing link structure to emulate the functioning of a WDM link. In either case the question arises, why should you integrate you code into *ns*, instead of implementing a proprietary simulation? A simple answer would be: to *REUSE* existing *TESTED* code. *ns-2* code has the advantage of being tested by hundreds of researchers worldwide, and hence the implementation of *ns-2* has evolved to be robust and reliable. The modules that implement the diffserv architecture at a node, viz., classifiers, meters, markers, shaper/droppers, are independent of the type of link on which packets are forwarded. Also *ns-2* supports a myriad of implementations of queuing mechanisms, traffic generators, all of which help a researcher to concentrate on the problem at hand, rather than worrying about the nitty-gritty implementation details of other required networking components.

Another important point is that *ns* is being used extensively by many researchers all over the world, who do not have to learn a new entire simulator from level zero to extend it. As we shall see, the WDM component has been designed such that newer scheduling algorithms and queuing mechanisms can be integrated into the WDM component with relative ease.

6.2 Nortel's Diffserv implementation in *ns-2*

Nortel's implementation of DiffServ provides the components that implement the DiffServ architecture in *ns*. It abstracts the functionality of a DiffServ router into a queuing mechanism. It uses multiple RED-based [18] queues to enable differential treatment of different traffic aggregates. Each RED queue consists of up to three virtual queues, one for each drop precedence. Different RED parameters are used for these virtual queues to enable differential treatment of traffic within a single class. It has two modules that correspond to the edge and core router responsibilities.

The edge queue class is responsible for maintaining multiple physical and virtual queues and processing those queues according to their parameters. Additionally it marks packets with code points and polices traffic aggregates. A policy determines the treatment that a traffic aggregate will receive at the edge device. Edge devices use policy information to determine with what code point to mark packets. Each policy defines a policer type, a target rate, and other policer-specific parameters.

Each traffic aggregate has an associated policer type, meter type, and initial code point. The implementation currently supports the following policer/meter types:

* TSW2CM [19] - time sliding window two color marker
* TSW3CM [20] - time sliding window three color marker
* TokenBucket [21] - token bucket
* srTCM [22] - single rate three color marker
* trTCM [23] - two rate three color marker

The implementation currently support the following scheduling mechanisms between queues:

* RR - plain vanilla round robin (default)
* WRR - weighted round robin
* WIRR - weighted interleaved round robin
* PRI - priority queueing

The core queue class emulates the core router in the DiffServ architecture; thus, it is intended to work downstream from an edge router. It forwards packets according to the marking done on them by the edge router.

6.3 Design and Implementation of the WDM Scheduler

To test the performance of our schedules the WDM Scheduler simulation was integrated into the network simulator (*ns-2*). This section is organized as follows: we discuss the network model and how we have mapped this model into an *ns-2* topology. We then discuss the simulation architecture, the main components involved in the simulator and explain the extensions required to the Nortel's DiffServ implementation. We have used the DiffServ module since we were trying to integrate and evaluate the performance of scheduling algorithms that schedule a differentiated mix of traffic. However our simulator has been designed to be highly extensible, and a subsection explains how different scheduling algorithms (for single hop WDM networks) and different queuing mechanisms could be incorporated in to our simulator. Each subsection explains any exported Tcl commands and demonstrates its usage.

Network model and simulation architecture

Recall that a broadcast and select WDM network is made up of N nodes interconnected by a passive star coupler that supports a set of C channels. The nodes are connected to a passive star coupler that couples the data transmitted on each of the C channels so that all the receivers receive data on every channel that was carrying data. In any practical network configuration we have that $N > C$ (see Figure 3.1). In a TT-FR configuration tunability is required only at the transmitters. The receivers are assigned a home channel and any data destined to a node should be transmitted on its home channel.

In an unfolded representation of the system (see Figure 3.1), two different nodes represent the transmitter and receiver at a node. Sets of nodes assigned a common home channel receive all the packet transmissions on their home channel. Hence having each of the N nodes connected to C LANs can represent the system model.

Network simulator (*ns-2*) does not have support for multi-channel links between nodes; neither does it support WDM links. We are primarily interested in the collapsed total traffic matrix since the scheduling algorithms works on the $N \times C$ collapsed total traffic matrix, T. The collapsed traffic matrix lends itself to the representation of the system model by an $N \times C$ mesh of unidirectional links. This *ns-2* network topology that effectively maps the system model described above, leads us to the architecture of our simulation, shown in Figure 6.1.

The topology of the simulation represents a single-hop WDM network, with N nodes, C channels and with tunable transmitters and fixed receivers. Also it can only model systems with collision-less scheduling algorithms.

The *WDMScheduler* object is the main component of the simulator architecture; it holds all the necessary data required for the functioning of the simulator. It holds the

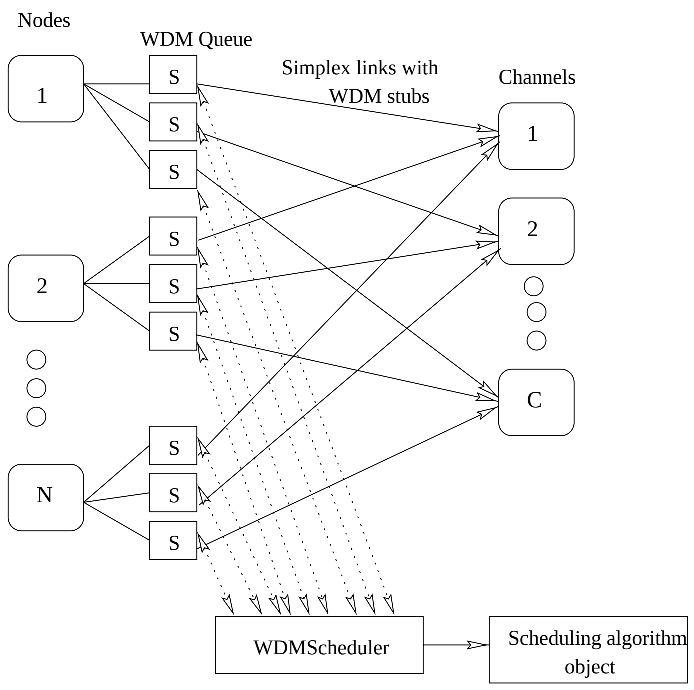

Figure 6.1: Simulation architecture

following data structures:

- References to all the *WDMQueue* objects that are placed before the WDM links and are registered with the WDM scheduler. It maintains a mapping from *ns* node id to internal indices to be used as node and channel indices of the traffic matrix and by the schedule computation object.

- A reference to the schedule computation object.

- A collapsed traffic demand matrix. The EF and AF slot demands are associated with corresponding node, channel for scheduling the requested number of EF, AF packets per schedule cycle.

- The actual schedule as computed by the schedule computation object, which forms the input to the slot time handler.

- References to the objects that maintain per node per channel per code point delay statistics and per channel per code point channel utilization statistics. Hence it needs to know the list of code points being used in the simulation.

- The physical queues that are configured to queue EF, AF and BE packets. This information is used to choose the queues from which to dequeue the next packet.

Further, the WDMScheduler class implements all the functionality required for the appropriate initialization and manipulation of these objects. The *WDMScheduler* object should be created during simulation initialization, which should be followed by setting certain parameters, namely,

* *channelBW_* and *bytesPerPacket_* - the channel bandwidth and the packets size (in bytes), are required to calculate the minimum slot time and calculating the channel utilization.

* *slottime_* - the slot time should be at least equal to the time it takes to transmit one packet. The slot time could have been computed given that both the channel bandwidth and the packet size are known, however keeping it configurable allows the flexibility of modeling the tunability characteristics of the transmitters.

* *expectedSchLgt_* and *slTolerance_* - the expected schedule length needs to be known; firstly to allocate best-effort traffic given the reservations for other traffic aggregates. And more importantly, to keep the delay and delay jitter bounded. Since the actual schedule length as computed by the scheduler may be greater than the expected schedule length a tolerance value (percentage of expected schedule length) needs to be specified.

The object that computes the schedule needs to be created and installed before packets can be scheduled; later subsections describe in detail how a WDM schedule is incorporated into *ns*, how the schedule computation object is installed and how newer scheduling algorithms can be included into the simulator. Tcl scripts drive the schedule computation, so that whenever the reservations change a schedule computation is initiated. When schedule computation is requested the current schedule is deleted, the schedule computation object computes the new schedule based on the current traffic demand matrix, and the slot time handler is scheduled. Once the schedule is computed, the schedule length is passed to the channel decomposition object[1] which computes the new indices into the schedule.

[1]The command *install-chan-decomposer* installs the channel decomposition object. The object should be installed before schedule computation is initiated for channel decomposition to take effect.

The *WDMQueue* objects are created by the link creation routines. Every *WDMQueue* object thus created should be registered with the *WDMScheduler*. The WDMScheduler expects certain functionality from any *WDMQueue* implementation; the abstract class *WDMSchedulerStub* abstracts this required functionality. The *WDMQueue* implementation is discussed in the next subsection.

Before we delve into the details of the WDM Queue implementation, let us discuss about the link structure in *ns*. A simple link is built up from a sequence of connectors. *ns* provides the instance procedure *simplex-link* {} to create a unidirectional point to point link from one node to another. The following describes the syntax of the simplex link creation:

$ns simplex-link <from> <to> <bandwidth> <propagation-delay> <queue-type>

The command creates a link from node <*from*> to node <*to*>, with specified <*bandwidth*> and <*propagation-delay*> characteristics. It further creates and installs a queue of type <*queue-type*> for the link. As shown in Figure 6.2, five instance variables define a link: *head_, queue_, link_, ttl_* and *drophead_*. Of interest to us are the *queue_* and *link_* variables, the *queue_* holds the reference to the queuing object and the *link_* holds the reference to the object that models the transmission and propagation delays of the link.

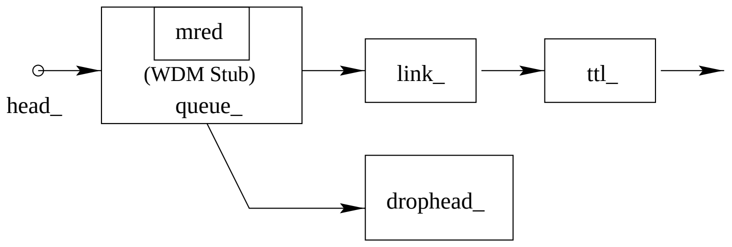

Figure 6.2: Components of a Unidirectional Link in *ns-2*

A reference of each of the simplex links created is stored in the global Tcl variable *link_*, which is an associative array, with <*from_id:to_id*> as the key. (This is the reason why we cannot have multiple point-to-point links between any two given nodes.)

WDM Queue structure

The *WDMQueue* object is the actual queue object that is placed before the link. The *WDMQueue* object receives the packets that the node generates to be transmitted on the associated channel. The primary responsibilities of the *WDMQueue* apart from the queuing mechanism are:

- enque packets that are received by the queue; for our topology it means queue the packets that are generated by the node to be forwarded on the associated channel

- ensure that packets are not deque'd at any other time apart from the time slot boundaries; this means that packets should be deque'd only by the time slot handler

- implement the basic functionality as expected by the *WDMScheduler*

The simulator component was designed with the view that the *WDMQueue* functionality should not be tied to any underlying queuing mechanism. The abstract class *WDMSchedulerStub* lists the interface between the *WDMQueue* and the *WDMScheduler*. This interface associates certain generic functionality to a WDMQueue:

* getNextPacket - deque the next packet from the queue according to the appropriate dequeuing mechanism. Also update the objects that maintain the associated delay and channel statistics

* fwdPacket - since the queue is also a connector the implementation of this method should forward the packet to its *target_*

* other methods that are used to pass data to and forth from the *WDMScheduler* object.

The class hierarchy of a WDMQueue that provides the functionality of an edge router of the DiffServ architecture is shown in Figure 6.3. The dsREDQueue class implements the queuing mechanism, and the modEdgeQueue contains the extensions to the Nortel's DiffServ implementation as discussed in a later subsection. The WDMQueue class provides a concrete implementation of each of the pure virtual functions declared in the WDMSchedulerStub class.

ns can be directed to create an object of the *WDMQueue* class and associate it with the link. The queue object should then be registered with the *WDMScheduler*, with the node id and channel id as parameters. The *WDMScheduler* then installs the node id and the channel id in to the node and channel map respectively, to associate node and channel indices to them. The *WDMScheduler* passes required information to the *WDMQueue* (as an instance of *WDMSchedulerStub*) and then stores a reference to the WDMQueue. The following snippet demonstrates the above:

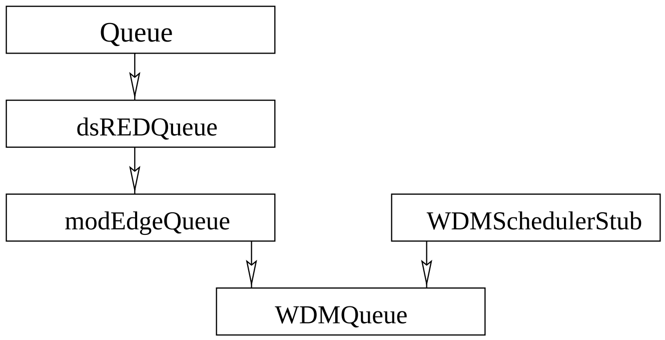

Figure 6.3: Class hierarchy of a WDM Queue

1. *$ns simplex-link $node $chan $bandwidth $prop_delay dsRED/WDMEdge*
2. *set que [[$ns link $node $chan] queue]*
3. *$wdmsch addQ $que [$node id] [$chan id]*

Line 1,3 drive the actions that were mentioned above, while line 2 get the reference to the queue object that was created during link creation. ($wdmsch holds the reference to the *WDMScheduler* object.)

Integrating other queuing mechanisms

As mentioned above the design of the *WDMQueue* does not tie it to any queuing mechanisms. The implementation of the *WDMQueue* above which extended the DiffServ edge router functionality, provides an example of how a queuing mechanism could be extended to incorporate it in to the simulator. A class that extends the basic queuing mechanism and implements the *WDMSchedulerStub* interface qualifies to be used as a WDM Queue.

Ideally the integration should not require changes to the existing code, neither should it require recompiling of the existing code base. However due to the way C++ implements casting of a multiply inherited object, the following changes need to be done: Suppose you define the class in file "newWDMQueue.h" and name the class *newWDMQueue*. You will have to include "newWDMQueue.h" in *~ns/wdmsched/WDMScheduler.cc*[2] and cast the object returned by *TclObject::lookup()* during the "addQ" command processing in the *WDMScheduler::command()* method.

[2]The directory ~ns refers to the directory that contains *ns-2* C++ code

Incorporating a WDM schedule into *ns-2*

A schedule needs to be computed before the WDM scheduler can start scheduling packets. The traffic demands from each node towards each channel have to be translated to the number of slots required per schedule cycle. The traffic demands for different classes of traffic are aggregated and slots are allocated to best effort traffic. For example, if your simulation is configured with EF, AF and BE sources, the traffic demands for the EF and AF sources is converted to slot requirements and then these demands are aggregated. This is then followed by slot allocation for BE traffic to form a combined traffic matrix. For converting the bandwidth demands to slots and for allocating BE slots the expected schedule length needs to be defined, along with a tolerance value (since the actual schedule length could be greater than the expected schedule length). The scheduler that has been installed during initialization of the WDM scheduler then computes the schedule. If the actual schedule length is less than the expected length plus the tolerance, the demands can be met else the demands need to be changed and the process repeated all over again. The schedule computed in this manner is installed and will be used until the traffic demands change, when a new schedule is computed and installed.

The schedule could be computed locally or by a central scheduler. However, in our case since the simulation does not intend to test the signaling (control) protocol we have implemented the centralized approach.

At the beginning of each time slot, packets are transmitted by a slot time handler (object of class slotTimeHandler (see ˜ns/wdmsched/ WDMScheduler.{h,cc})) , according to the (possibly decomposed) schedule. The slot time handler executes every *slottime_* and does the following:

- For each channel
  - Identifies the node that has exclusive right to transmit a packet on that channel.
  - Requests a packet destined towards the channel to be deque'd from that node's queue. It does not instruct which traffic aggregate the packet should belong to; this information is local to the *WDMQueue* (refer to the discussion about the *WDMQueue* implementation for more details).
  - Forwards the packet to that channel.
- At the end of each schedule cycle, instructs each WDM Queue to reinitialize its state

and starts from slot number 0.

- Schedules the next event to occur after a time offset of "slottime_"

Only the slot time handler should forward packets. The implementation of packet forwarding in *ns*, attempts to forward a packet every time a new packet arrives, as well as every time the link delay object completes the transmission of the packet currently being transmitted. To ensure that only the slot time handler forwards packets, the queue is always maintained in the "blocked" state (see ~ns/queue.{h,cc}). Also the queue handler parameter (object of class QueueHandler) sent by the queue should not be used for the callback at the completion of a packet transmission, thus ensuring that the queue remains in the blocked state.

Integrating scheduling algorithms into the scheduler

The WDM single hop simulator component was designed such that scheduling algorithms could be independently implemented and integrated into the scheduler, without requiring any changes to the central scheduler implementation. We have demonstrated this by integrating two of our algorithms into the model.

The central WDM scheduler (see ~ns/wdmsched/ WDMScheduler.{h,cc}) references to an object that implements the WDM scheduler interface (see ~ns/wdmsched/ WDMSchedulerInterface.{h,cc}). The WDM scheduler interface defines the functionality that it expects from any schedule implementation. The object implementing the scheduling algorithm is instantiated through Tcl scripts and should be registered with the central scheduler before schedule computation can begin. We use the implementation of the optimal schedule computation object as an example of the above-described procedure. Figure 6.4 shows the class hierarchy of the optimal schedule computation object (see ~ns/wdmsched/ OptimalScheduler.{h,cc}):

The WDM scheduler interface defines a method "getSchedule" that takes a traffic matrix and returns a schedule computed by the scheduling algorithm. Since the OptimalScheduler extends the WDM scheduler interface, it is also a TclObject and hence can be created at simulation runtime by Tcl scripts as shown below:

1. *set wdmsch [new WDMScheduler $nodes $chan]*
2. *set optimal-scheduler [new WDMScheduler/Optimal]*

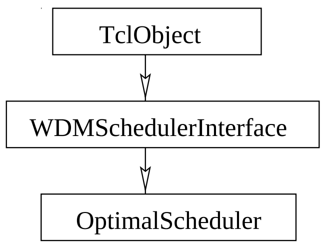

Figure 6.4: Class hierarchy of the optimal scheduler

3. *$wdmsch install-scheduler $optimal-scheduler*

The tcl code above is self-descriptive; the WDM scheduler computes the schedules on "compute-schedule" command. The design also makes it possible to change the scheduler at runtime, just in case one wishes to experiment with multiple schedulers within a single simulation run. The only requirement is to create an object of the class that implements the desired scheduling algorithm, and register it with the WDMScheduler object. On the next "compute-schedule" command, the schedule is computed using the new scheduling algorithm.

Our simulator provides two schedule algorithm implementations:

- Optimal algorithm based on the preemptive open shop scheduling algorithm [26]. The object that implements this algorithm can be instantiated by creating an object of class *WDMScheduler/Optimal*
- The scheduling heuristic proposed by Sivaraman [27]. The object that implements this algorithm can be instantiated by creating an object of class *WDMScheduler/Vijay*

Statistics gathering

The statistics of interest are: delay, delay jitter, channel utilization and number of packet drops; for each of the traffic aggregates competing for the resources, namely, buffer requirement and channel bandwidth. Of these delay, delay jitter and packet drops need to be maintained per node per channel per traffic aggregate, whereas channel utilization (alternatively channel throughput) is monitored per channel per traffic aggregate. Objects of class WDMDelayStatistics maintain queuing delay and associated delay jitter and objects of class WDMChannelStatistics maintain the statistics required for channel utilization calculation. The queue implementation maintains the packet drop count, in our case the dsREDQueue maintains the number of early drops and normal drops per code point (or traffic aggregate).

Delay/Delay Jitter calculation: When a packet arrives at a node or is generated by the node itself, it forwards the packet to the WDM queue. The WDMQueue::enque method records the packet's arrival time in that packet's common header, in the timestamp (ts_) field. Later when a packet is deque'd from the WDM Queue and before it is forwarded to the next object on the link hierarchy, the queuing delay is calculated and the WDMDelayStatistics' object is informed about it. The stats object uses this information to calculate the mean queuing delay and the standard deviation of the samples from the mean value. Similarly it also calculates the mean and standard deviation for the delay jitter. The WDM scheduler has exported the following commands to fetch these values from a tcl script: *get-mean-delay, get-delay-variance, get-delay-jitter, get-mean-delay-jitter* and *get-delay-jitter-variance*; each of these commands take the node id, channel id and the code point as parameters.

Calculation of channel throughput/channel utilization: Objects of class WDM-ChannelStatistics maintain the number of packets that have been transmitted on the channel associated with that object. Since every packet is assumed to be of the same size, only number of packets transmitted is counted (a byte count would over run relatively faster than packet count for longer simulation runs). Every time a packet is transmitted, by any of the nodes on the channel, the packet count is incremented. The WDM scheduler has exported the following commands to fetch the channel utilization and throughput values from a tcl script: *get-channel-utilization* and *get-channel-throughput*; each of these commands expects a channel id and a code point as input parameters. Channel utilization is defined as the ratio of bandwidth used to the channel bandwidth.

Note: The WDM scheduler needs to know the packet size and channel bandwidth during initialization. This is done by setting the bound variables: *channelBW_* & *bytesPerPacket_*.

Packet drops: These are accounted by the queuing mechanism, see the WDM Queue implementation for the functions that need to be implemented by a WDM Queue. The number of early drops, normal drops and total number of drops can be fetched from the WDM scheduler by the following commands: *get-edrops, get-drops* & *get-total-drops*; each of the commands expect node id, channel id and code point as parameters.

Extensions to Nortel's DiffServ implementation

Nortel's DiffServ implementation classifies packets based on the source and destination addresses carried by the packets. Specifically this implies that a <source, destination> pair uniquely identifies a policy entry. A policy entry is added by the following command:

$que addPolicyEntry <src id> <dst id> <policerType> <initial code point> <policer parameters>

What this means is that all packets originating at a source and destined to the same destination experience the same quality of service. Clearly one might want to set up multiple flows having the same *<src, dst>* but experiencing different treatment. For example, there may exist a audio stream flowing between source s and destination d and also a telnet connection between the two. Obviously these two connections have different delay and throughput requirements and hence should experience different treatment. Hence we need a additional field in the header that could identify the type of packet, and we have used the flow id field of the IP header (see ~ns/ip.h). Now the tuple *<src, dst, flowid>* is used to identify a flow and hence a policer entry in the PolicyTable. Since it has been our intent not to modify any of the files of the *ns* distribution, flow id was introduced in the policy table entry by extending the *struct policyTableEntry* (see ~ns/dsPolicy.h). Similarly we also extended the *class PolicyTable* to include a table of modified policy table entries, and also to associate each policy table entry to a *<src, dst, flowid>* tuple (see ~ns/wdmsched/ modDSPolicy.{h,cc}).

In the DiffServ architecture only an edge router performs packet classification and policing. Hence only the *edgeQueue* class (see ~ns/edge.cc,h) maintains a policy table. Hence a new edge queue class had to be implemented that extends the dsREDQueue (see ~ns/dered.cc,h) and maintains a policy table with the modification mentioned above (see ~ns/wdmsched/ modDSEdge.{h,cc}). Apart from the fact that this queue maintains the modified policy table, its other functionality remains the same as *edgeQueue* class. An instance of this modified edge queue could be incorporated into the link creation in the usual way by:

$ns simplex-link <src id> <dst id> <bw> <prop_delay> dsRED/modEdge

This queue could be now used in building a DiffServ enabled WDM single hop network as described in preceding subsections.

Emulating EF PHB

Nortel's diffserv implementation does not tie code points to PHBs, as suggested by the RFCs. It gives us the freedom of designing our experiment configuration by using as many code points as required and experimenting with the various possible combinations of the policer and schedulers. As mentioned above the implementation supports the following policer types: *TSW2CM, TSW3CM, TokenBucket, srTCM, trTCM.* Each policer has a corresponding meter type associated with it. The implementation also provides the following mechanisms for choosing the queue from which to transmit the next packet: *RR, WRR, WIRR, PRI.* (refer to [7] for the policer parameters and related discussion).

However, the implementation does not specify which combination of the policing/metering and scheduling mechanisms yields a certain behavior. Hence it is a matter of experimentation to determine the combination that yields the EF PHB. We found the following configuration parameters emulate an EF PHB:

1. *$que addPolicyEntry <src> <dst> <fid> TokenBucket 10 <cir> <cbs>*
2. *$que addPolicerEntry TokenBucket 10 12*
3. *$que addPHBEntry 10 0 0*
4. *$que addPHBEntry 12 0 1*
5. *$que configQ 0 0 <min_th> <max_th> 0.0*
6. *$que configQ 0 1 0 0 1.0*

In the snippet above, line 1 associated <*src, dst, fid*> tuple to a token bucket policer, sets the initial code point and sets the token bucket parameters (the committed information rate (CIR) and committed burst size (CBS)). Line 2 instructs the policer to downgrade the packets carrying code point 10 to 12 if they are out-of-profile. Line 3 and 4 instruct the queue to add the packets carrying code point 10 to physical queue 0 and virtual queue 0 (also know as drop precedence). And to add packets with code point 12 to physical queue 0 and virtual queue 1. The lines 5 and 6 set the RED queue configuration parameters. The parameters on line 5 direct the queue not to drop any in-profile packet and those on line 6 direct it to drop all out-of-profile packets. Thus emulating the EF PHB.

The next chapter presents delay-related performance measures for guaranteed traffic in a single hop WDM network, using the WDM simulator component just described.

Chapter 7

Numerical Results

We presented the results related to channel throughput attainable by the scheduling algorithms and their schedule lengths in Chapter 4.5. In this chapter we turn our attention to results related to scheduling delays experienced by guaranteed service traffic. All the results presented in this chapter were obtained using the WDM component we implemented on *ns-2*. We measure the average scheduling delay, variance[1] in delay and delay jitter statistics for each scenario, since we feel that together they characterize the delay-related performance of the scheduling algorithms. We are interested only in the scheduling delays packets experience, that is, we measure the time a packet spends in the queue waiting to be scheduled for transmission. In the following discussion we use the term delay to mean scheduling delay.

We have studied the effect of nonuniform traffic on the schedule lengths and the channel throughput in Chapter 4.5, in this chapter our primary experiments are based on uniform traffic distribution. Specifically, every node has the same traffic requirement on all the channels, and all the nodes share the bandwidth available on a channel equally. Nodes reserve a part of their share for guaranteed service traffic[2]. Since we consider a *wavelength-limited* system the number of nodes far exceeds the number of channels, this along with the uniform traffic distribution guarantee that both the scheduling algorithms presented in Chapter 4 compute optimal schedules. This means that the number of channels in the system does not affect the overall scheduling delays experienced by packets. Hence to speed

[1]We plot the standard deviation, instead of variance, in delay to keep the same units as that for delay/delay jitter.

[2]For our simulations we had CBR (constant bit rate) sources generating both the guaranteed and best effort service.

up the simulation run time we use smaller number of channels, 6 to be precise. However, what matters is the channel bandwidth. We present results for channel bandwidth of 1 Gbps and 10 Gbps.

We feel that guaranteed services will account for only a small percentage of the total available bandwidth, hence we consider reservations of 10% to 40%. In a WDM schedule, for that matter any TDM schedule, the scheduling delays are a function of the schedule lengths. By design of the simulation, we are guaranteed to obtain optimal schedules, hence the total demand (guaranteed service plus best effort) per node directly affect the delay packets experience. We experiment with demands per node of 10 to 40, that is, for a system with 200 nodes the schedule lengths would vary from 2000 to 8000 slots. We consider fixed-sized packets, each of size 1000 bytes. Table 7.1 summarizes the number of slot reservations per node, the total demand per node and the resulting schedule lengths that we used in our simulations. As the table shows we execute simulations for four slot demands with 10% to 40% reservation, resulting in sixteen combinations for each of the scenario defined below.

Demand per node (in slots)					Schedule Length				
Total demand	Reservation for guaranteed services				in slots		in milliseconds		
					N=200	N=20	N=200		N=20
	10%	20%	30%	40%			1 Gbps	10 Gbps	1 Gbps
10	1	2	3	4	2000	200	16	1.6	1.6
20	2	4	6	8	4000	400	32	3.2	3.2
30	3	6	9	12	6000	600	48	4.8	4.8
40	4	8	12	16	8000	800	64	6.4	6.4

Table 7.1: Reservations per node per cycle and resulting schedule lengths

Two sets of experiments were performed, one in which the guaranteed traffic was non-backlogged and in the other the guaranteed traffic was backlogged. In the non-backlogged case the guaranteed traffic sources sent packets exactly at the committed information rate (CIR). Further, best effort traffic is assumed to be always backlogged. Within each case we had two sets of experiments one in which the schedule used to transmit packets was exactly as generated by the scheduling algorithm. And for the other set, the schedule was first subject to channel decomposition (discussed in Chapter 5), that is, once the schedule length (M) is obtained channel decomposition is applied to the indices of the schedule

0 to $M-1$ and the resulting indices are then used to index the schedule. As discussed in Chapter 5 channel decomposition distributes time slots that otherwise would be adjacent evenly across the schedule. As mentioned above, we consider channel bandwidths of 1 and 10 gigabits per second (Gbps).

In the differentiated services architecture [13] guaranteed service traffic is policed by approprite policers. Our simulations consider a traffic mix of EF [14] and best effort services. EF sources are normally policed by a Token Bucket policer. The token bucket policer is parameterized by a committed information rate (CIR) and committed burst size (CBS), that is, a token bucket policer considers a burst packets, of size equal to the CBS, as in-profile. We consider the effect of this committed burst size parameter of the token bucket policer on the delay experienced by packets. The scenarios are summarized in Table 7.2.

Traffic	Channel Decomposition	Channel bandwidth	Number of Nodes	CBS
Non-backlogged	no	1 Gbps	200	1-12
			20	1-12
	yes	1 Gbps	200	1-12
			20	1-12
Backlogged	no	1 Gbps	200	1-12
			20	1-12
		10 Gbps	200	4,6,8,10,12
	yes	1 Gbps	200	1-12
			20	1-12
		10 Gbps	200	4,6,8,10,12

Table 7.2: Simulation scenarios.

We run simulations for $1,000,000$ time slots for channel bandwidth of 1 Gbps and for $5,000,000$ time slots for channel bandwidth of 10 Gbps, corresponding to 8 and 4 seconds of simulation time respectively. Each point on the graphs corresponds to the average of the corresponding parameter at all the nodes on all the channels. For instance, a point on the graph shown in Figure 7.1, corresponds to the average of the scheduling delay experienced by the packets at each of the 200 nodes on each of the six channels.

Graphs shown in Figures 7.1-7.3 and 7.7-7.9 plot average delay, delay jitter and standard deviation of delay, against the *committed burst size* (CBS) granted to the guaran-

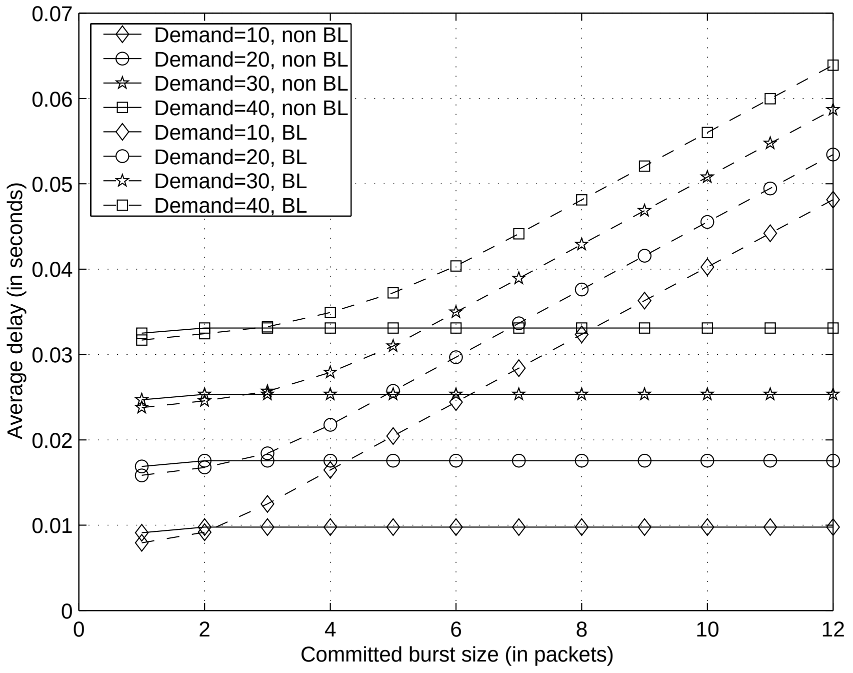

Figure 7.1: Average scheduling delay comparision for non-backlogged and backlogged traffic on a 1 Gbps channel with 200 nodes, for a reservation of 40%

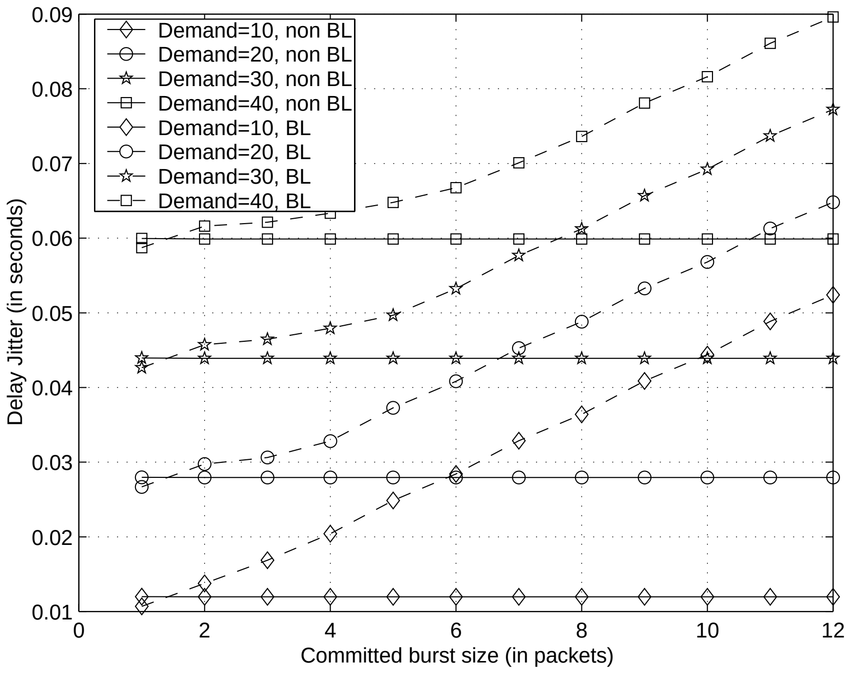

Figure 7.2: Delay Jitter comparision for non-backlogged and backlogged traffic on a 1 Gbps channel with 200 nodes, for a reservation of 40%

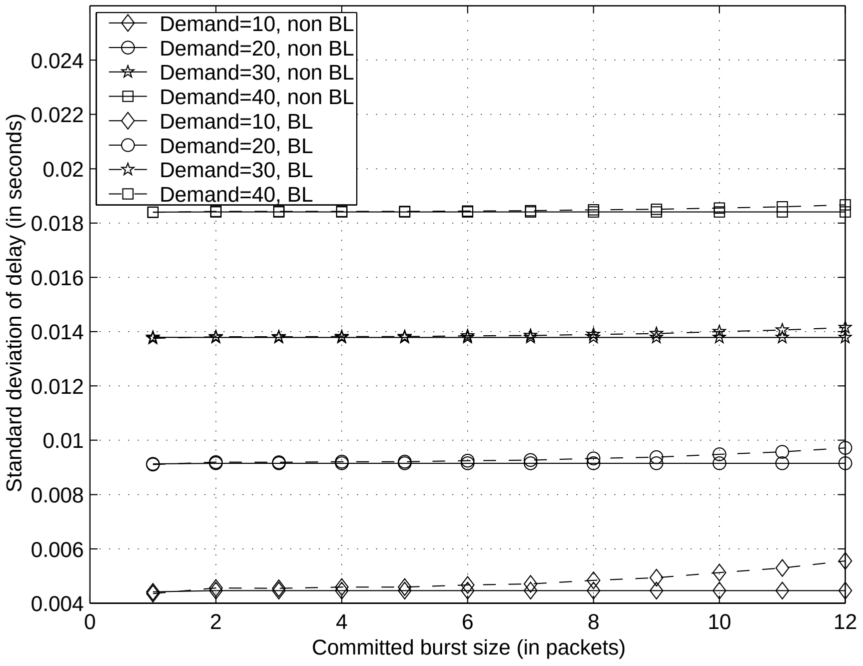

Figure 7.3: Standard deviation in scheduling delay for non-backlogged and backlogged traffic on a 1 Gbps channel with 200 nodes, for a reservation of 40%

teed traffic sources, for varying demands per node (equivalently, varying schedule lengths). CBS is represented in units of packets, hence a CBS value of 12 packets corresponds to $12 \times 1000 \times 8 = 96Kbits$. Figure 7.1 compares the average scheduling delay packets experience on a system that has non-backlogged guaranteed sources, with the delay experienced on a system that has backlogged guaranteed sources. It can be observed that in a non-backlogged system packets experience identical delays for all the values of CBS, this is due to the fact that in such systems the queues never build up. Delay increases with an increase in the schedule length, but it can be observed that delays are less than the length of a schedule. For instance, with a demand of 10 per node in a system with 200 nodes, the optimal schedule length would be 2000 slots, equal to 16ms on a 1 Gbps channel, and the average delay experienced is close to $10ms < 16ms$. The delay observed in this case would be the best case with a non-decomposed channel schedule. However, in the backlogged traffic case, the scheduling delay increases almost linearly with CBS. At lower values of CBS, lesser bursts are allowed restricting the queue build up and resulting in lower delays. For higher values of CBS packets accepted by the policer exceed the rate at which they are transmitted, hence queues build up and the resulting delays are no longer bound by the

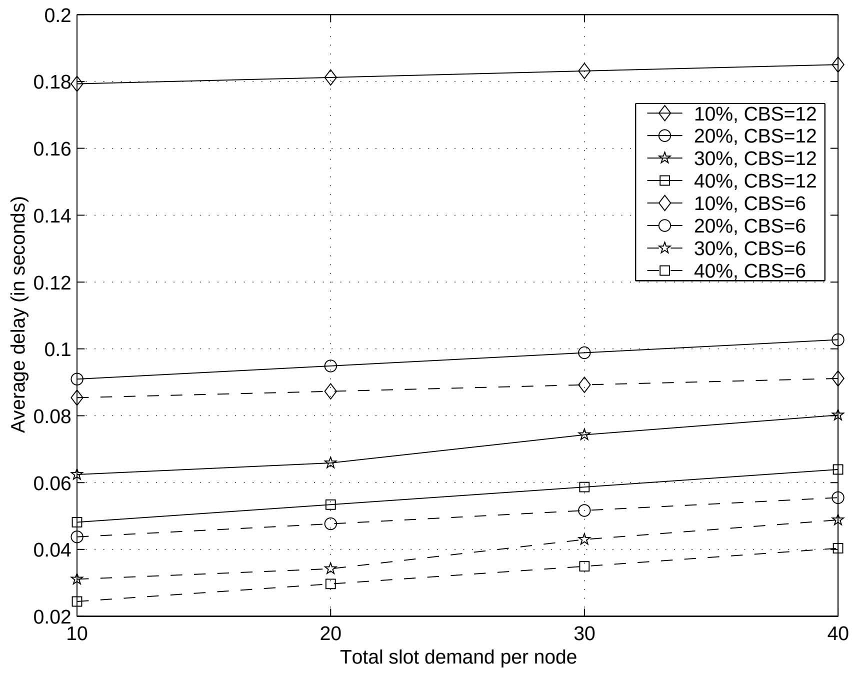

Figure 7.4: Average scheduling delay comparision for committed burst size of 6, 12 packets on a 1 Gbps channel with 200 nodes, for $10\% - 40\%$ reservation

schedule lengths.

Figure 7.2 compares delay jitter[3] in systems with non-backlogged and backlogged guaranteed traffic sources. Again the delay jitter experienced by the packets in a system with non-backlogged guaranteed traffic is identical for all the values of CBS. As with the average delay, the delay jitter is also bounded by the schedule length. Delay jitter in a system with backlogged guaranteed service traffic increases linearly, after a certain lower CBS value. It should be noted that the best case delay jitter values are greater than the average delays and are very close to the schedule lengths. This can be attributed to the fact that packets are transmitted in bursts, which reduces the average delay, however the delay jitter increases (see Chapter 5). Figure 7.3 compares the standard deviation of delay in systems with non-backlogged and backlogged guaranteed traffic sources. In both the cases, the higher the schedule lengths the higher the deviation. Systems with non-backlogged guaranteed traffic experience identical deviations for all values of CBS. In systems with backlogged guaranteed traffic the deviation slightly increases with increase in CBS.

[3]Delay jitter is calculated as the difference between the maximum and minimum delays experienced by packets.

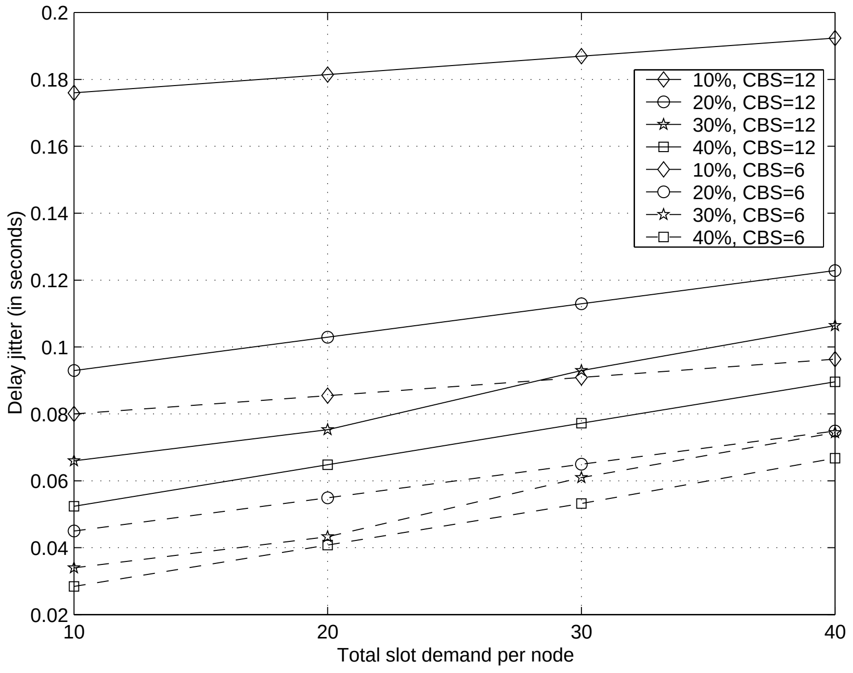

Figure 7.5: Delay Jitter comparision for committed burst size of 6, 12 packets on a 1 Gbps channel with 200 nodes, for 10% − 40% reservation

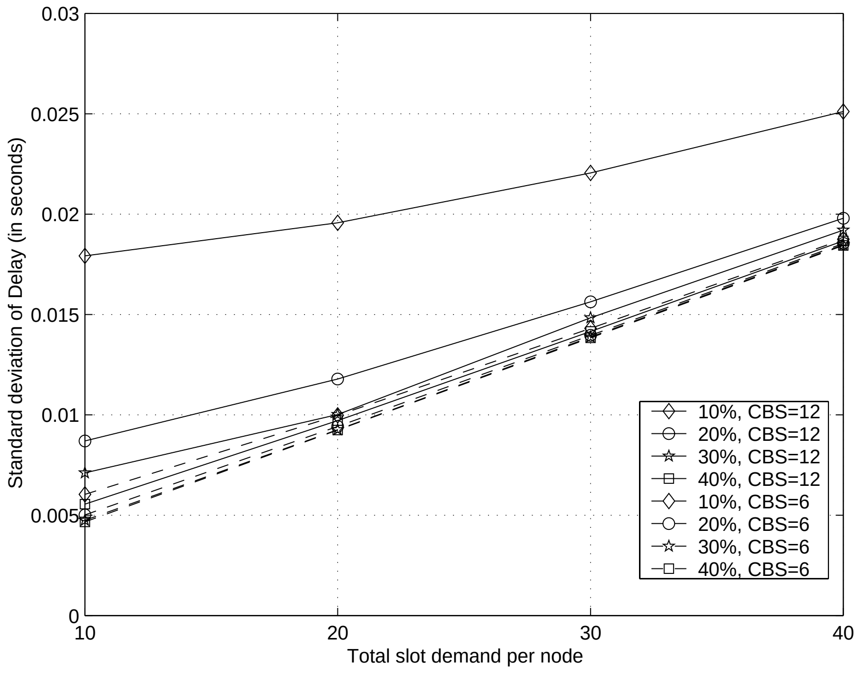

Figure 7.6: Standard deviation in scheduling delayfor committed burst size of 6, 12 packets on a 1 Gbps channel with 200 nodes, for 10% − 40% reservation

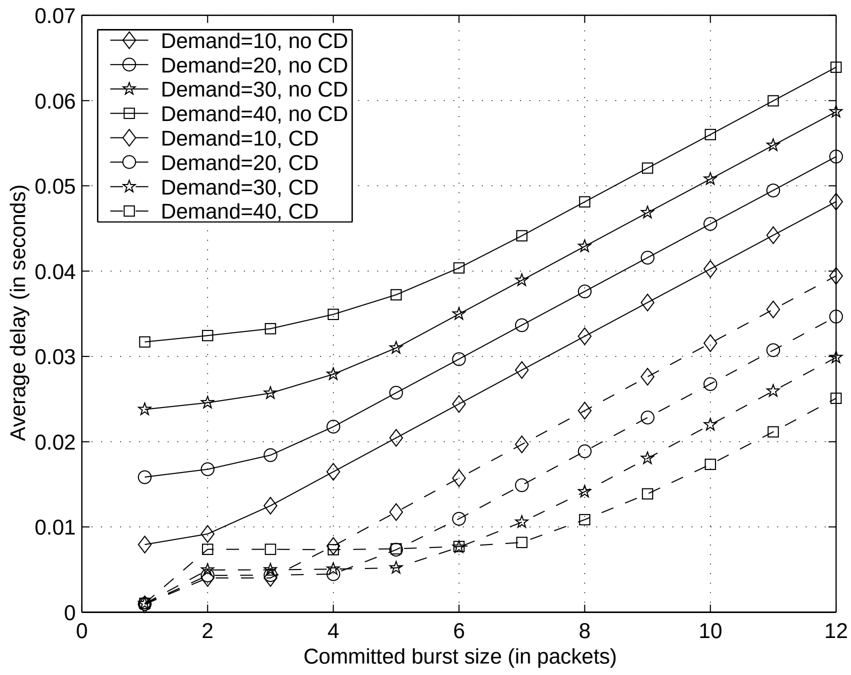

Figure 7.7: Average scheduling delay comparision with and without channel decomposition on a 1 Gbps channel with 200 nodes, for a reservation of 40%

Ideally, scheduling algorithms should decouple bandwidth and delay, that is, it should be possible for a traffic aggregate to obtain a smaller delay while still reserving only a small bandwidth. However, in the case of TDM-style schedules, where packet streams are scheduled in cycles, inherently bandwidth and delay are tied to each other. In Figures 7.4-7.6 we plot the average delay, delay jitter and standard deviation of delay for 10% to 40% reservation for varying demands per node. It was observed above that delay and delay jitter increase linearly with CBS, hence we use this opportunity to compare the delay-performance with a CBS of 6 and 12 packets (corresponding to 48Kbits and 96Kbits, respectively). For all the remaining simulations we only consider systems with backlogged guaranteed traffic sources.

Figures 7.4 and 7.5 confirm our intuition that higher reservation results in lower average scheduling delay and delay jitter. Also for a given reservation, the scheduling delay and delay jitter increase with an increase in schedule lengths. The delay and delay jitter for a burst size of 6 are considerably lower than that with a burst size of 12.

We now present results obtained after applying channel decomposition, as was described in Chapter 5. Channel decomposition results in adjacent slots being spaced well

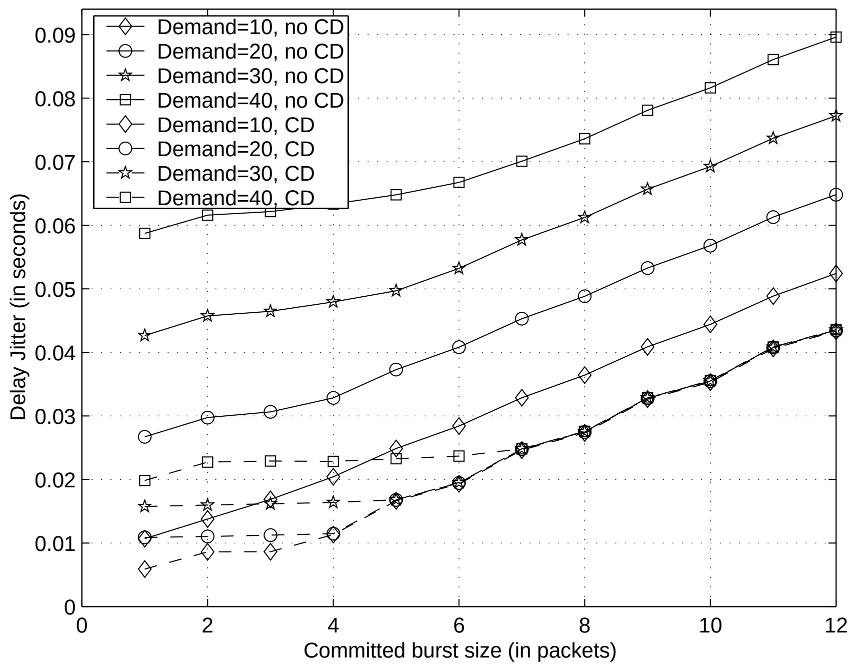

Figure 7.8: Delay Jitter comparision with and without channel decomposition on a 1 Gbps channel with 200 nodes, for a reservation of 40%

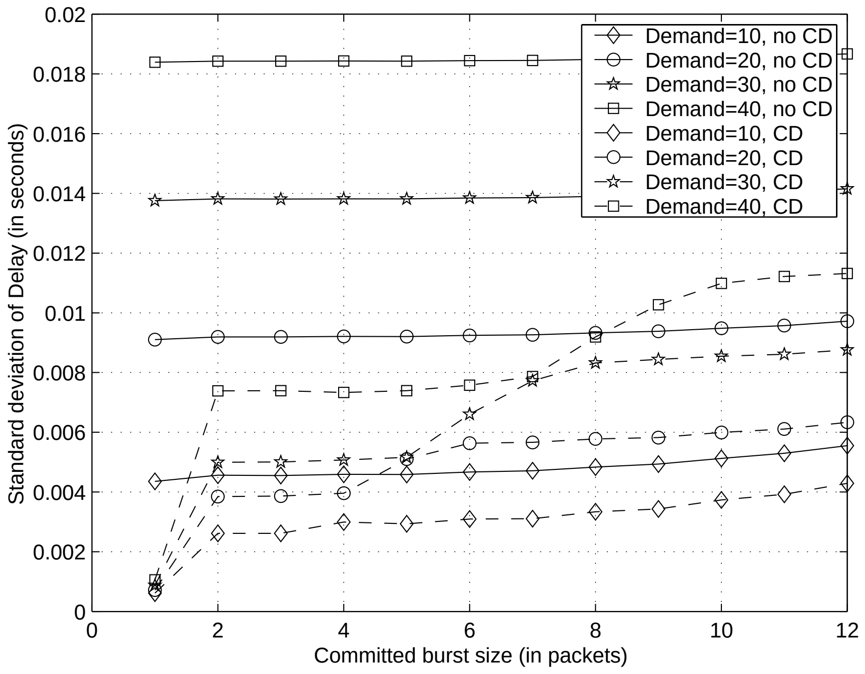

Figure 7.9: Standard deviation in scheduling delay with and without channel decomposition on a 1 Gbps channel with 200 nodes, for a reservation of 40%

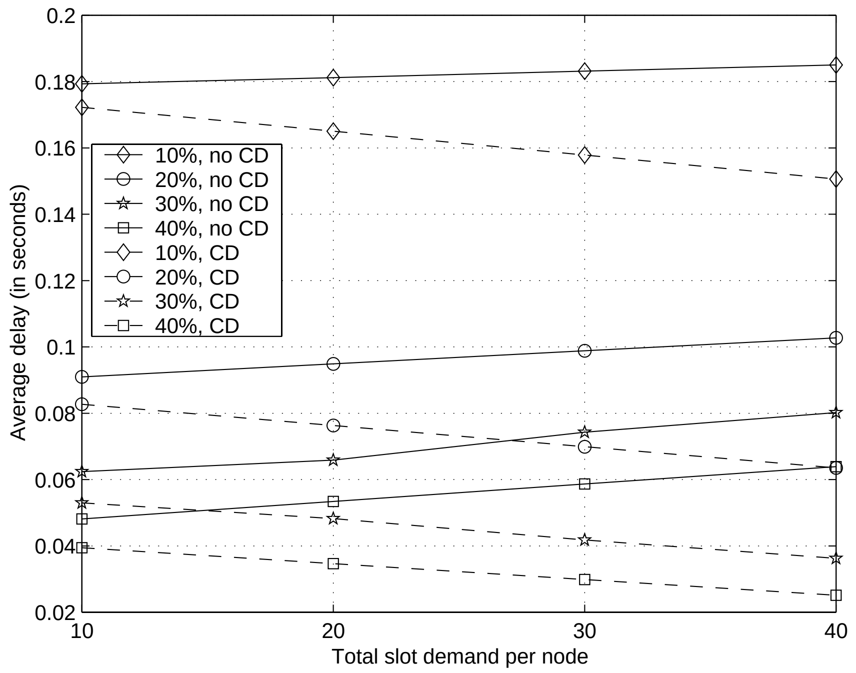

Figure 7.10: Average scheduling delay comparision with and without channel decomposition on a 1 Gbps channel with 200 nodes, for committed burst size of 12 packets

apart within the schedule. We expect this behavior to reduce the delay, delay jitter and standard deviation. Figures 7.7-7.9 plot the delay, delay jitter and standard deviation of delay, against CBS, for varying demands per node, with the schedules being used with and without channel decomposition. Figure 7.7 shows that the use of channel decomposition, in general, results in lower scheduling delays. And channel decomposition is more effective for higher schedule lengths. It can be seen that for higher demands per node the scheduling delays are identical for lower CBS values, and then increase linearly with CBS. Figure 7.9 shows that channel decomposition results in lower standard deviation of delay.

Figure 7.8 shows an interesting result, in that, for higher values of CBS, delay jitter experienced by packets served in longer schedules are equal to that experienced by packets served by smaller schedules. This means that with channel decomposition we can enjoy the flexibilility of having higher demands per node, which allows a higher granularity of reservations. For instance, a total demand per node of 10 slots facilitates the granularity of reservation to be 10%, with a total demand of 40 slots per node the granularity of reservation can be as low as 2.5%. With this flexibility different nodes can allocate differing percentages for guaranteed traffic sources.

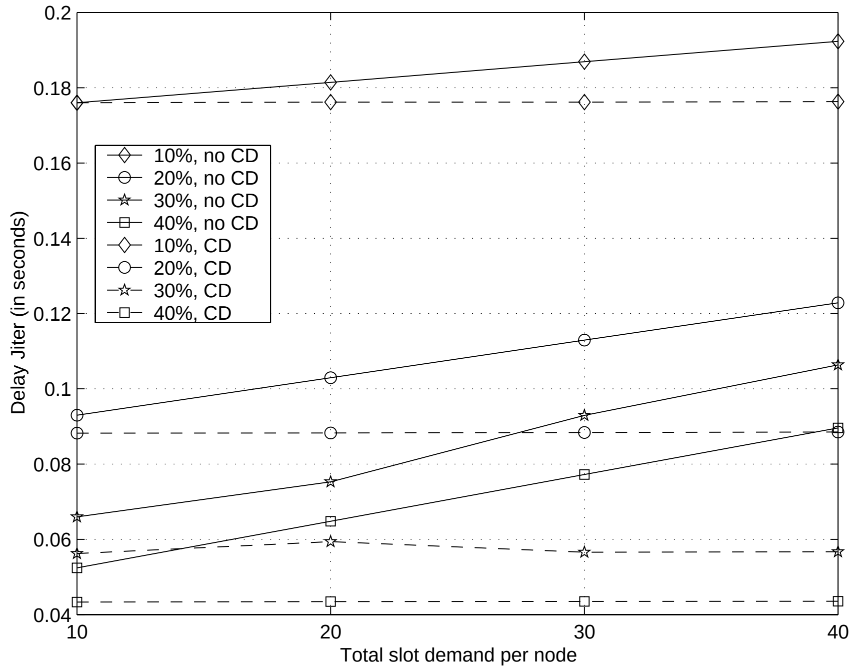

Figure 7.11: Delay Jitter comparision with and without channel decomposition on a 1 Gbps channel with 200 nodes, for committed burst size of 12 packets

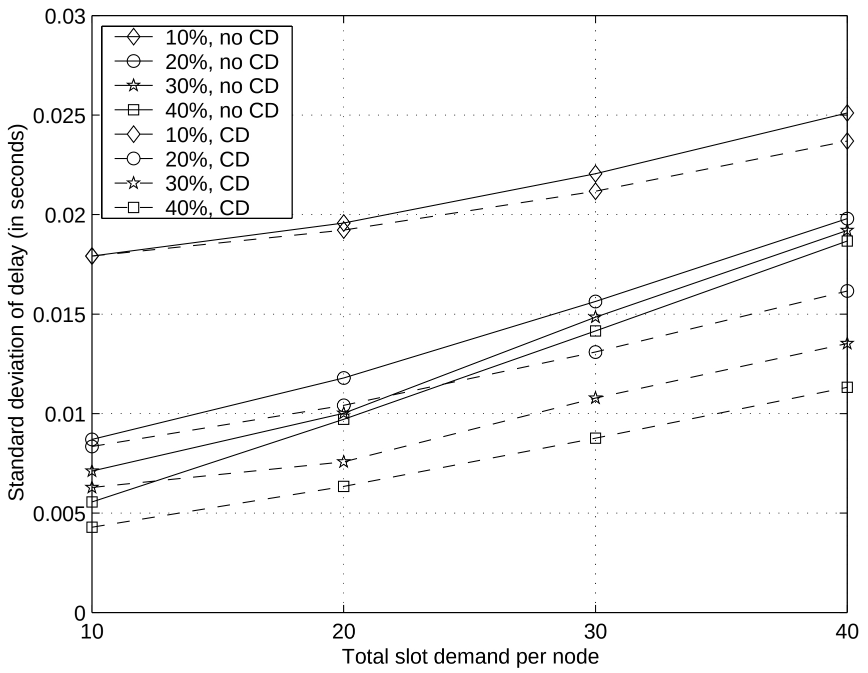

Figure 7.12: Standard deviation in scheduling delay with and without channel decomposition on a 1 Gbps channel with 200 nodes, for committed burst size of 12 packets

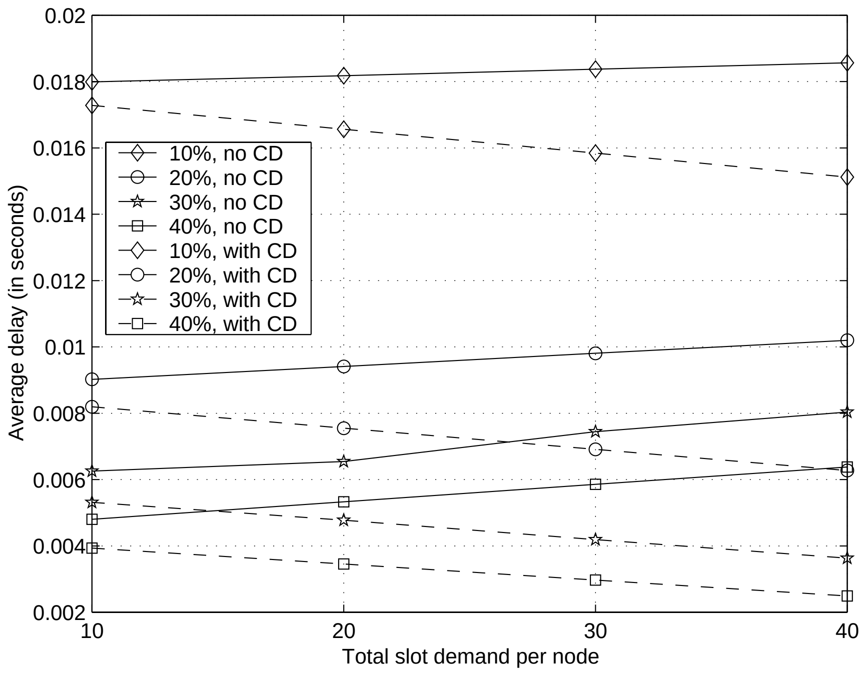

Figure 7.13: Average scheduling delay comparision with and without channel decomposition on a 10 Gbps channel with 200 nodes, for committed burst size of 12 packets

Figures 7.10-7.12 plot the delay, delay jitter and standard deviation of delay for 10% to 40% reservation for varying demands per node, with the schedules being used with and without channel decomposition. The results show that channel decomposition reduces delay (delay jitter, standard deviation) in general, and is more effective for longer schedule lengths and for higher reservations. All the results presented upto this point used a 1 Gbps channel. Figures 7.13-7.15 compare the performance of channel decomposition on a 10 Gbps channel, and we observe similar behavior as in Figures 7.10-7.12.

The next chapter summarizes our work and presents directions for future research.

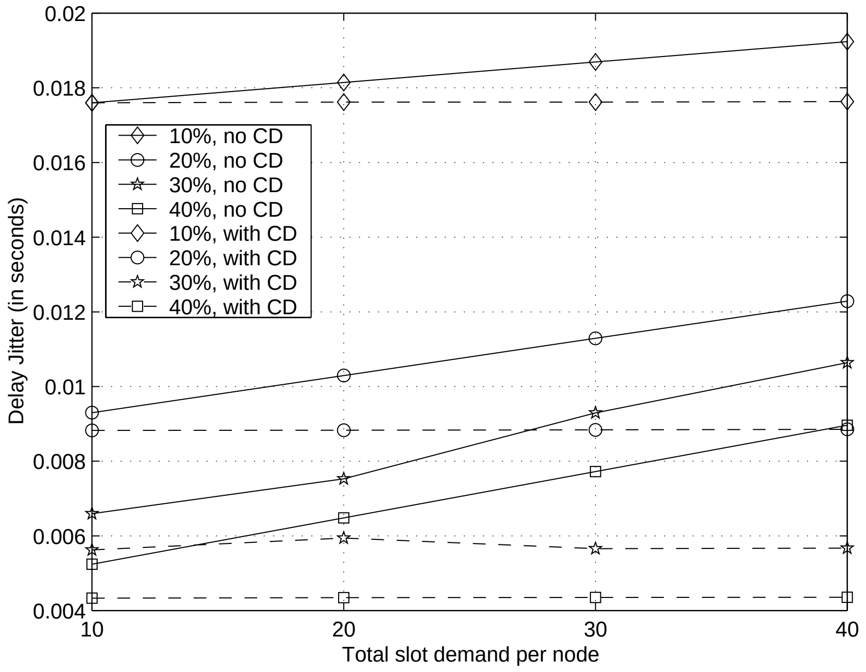

Figure 7.14: Delay Jitter comparision with and without channel decomposition on a 10 Gbps channel with 200 nodes, for committed burst size of 12 packets

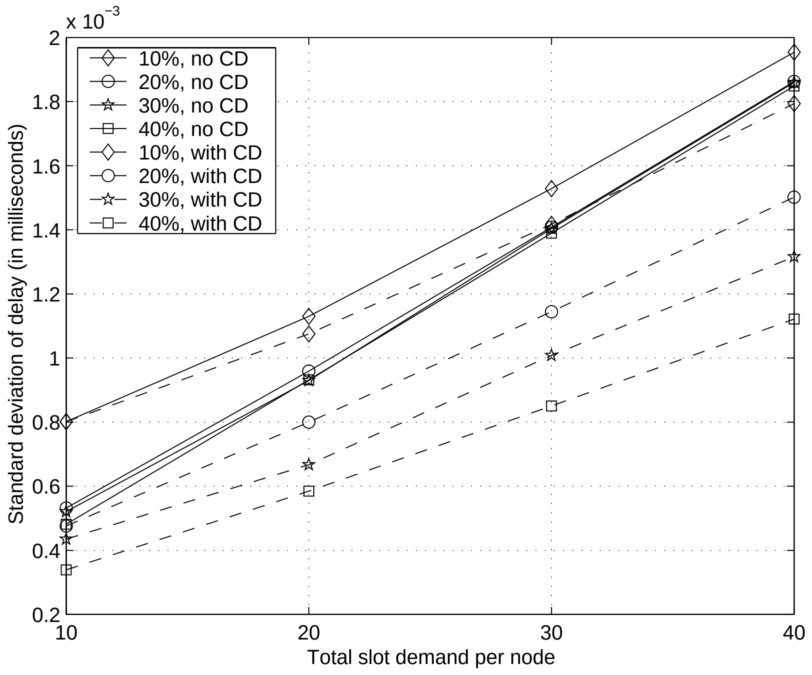

Figure 7.15: Standard deviation in scheduling delay with and without channel decomposition on a 10 Gbps channel with 200 nodes, for committed burst size of 12 packets

Chapter 8

Summary and Future Research

8.1 Summary

We have considered the problem of allocating slots to best effort traffic streams in a *max-min fair* manner, for arbitrary guaranteed service traffic demands in a broadcast single hop optical network. Our objective was to allocate excess (or a specified number of) slots to best effort traffic, to increase channel utilization, without considerably increasing the schedule lengths. We also wanted to study the effect of the scheduling algorithms on the delay-related performance measures, namely, average scheduling delay, delay jitter and delay variance. We showed that best effort allocation using the schemes we proposed, *BE-OS* with preemptive scheduling algorithms and *BE-OSTL* with nonpreemptive OSTL scheduling algorithms, increases the channel utilization without considerably increasing schedule lengths. Specifically, in the case of preemptive open-shop scheduling algorithm the *BE-OS* scheme increases the channel utilization to 100% without increasing the schedule length. Both the scheduling algorithms we considered have slightly higher running times, however, in practical IP DiffServ environment we expect reservations to be slowly varying. Hence it is reasonable to use the higher complexity scheduling algorithms to achieve higher channel utilization.

8.2 Future Research

The scheduling algorithms we considered serve traffic aggregates in TDM cycles. Due to packets being transmitted in schedules, delay is coupled with bandwidth reservations, that is, traffic aggregates would experience lower delay by reserving higher bandwidth. Ideally, however, aggregates should be able to demand lower scheduling delays while reserving lower bandwidth. Algorithms to schedule packets in a single hop WDM network need to be developed that can decouple delay and bandwidth.

In a single-hop WDM network having fixed receivers, the unicast and multicast traffic can be scheduled by a single scheduling algorithm, however, research needs to be done to identify and address the issues for the support of quality-of-service for multipoint communication in a broadcast WDM network.

Future areas for research to incorporate the schemes, we discussed, in the Helios optical access network testbed would be development of state machines for the protocols for distributing the global traffic matrix before best effort allocation. Also the protocols need to redesigned for efficient hardware implementation. Further, mechanisms need to be specified for aggregating traffic destined to different nodes receiving data over the same channel to isolate individual traffic aggregates (protection) and ensure fairness.

Bibliography

[1] DARPA Next Generation Internet (NGI) Program. http://www.darpa.mil/ito/research/ngi/index.html.

[2] Helios: Regional Testbed Optical Access Network for IP Multicast and Differentiated Services. http://www.anr.mcnc.org/projects/Helios/Helios.html.

[3] M. Kuznetsov et al. A next-generation optical regional access network. *IEEE Communications Magazine*, 38(1):66–72, January 2000.

[4] E. Hall *et al.* The Rainbow-II gigabit optical network. *IEEE Journal Selected Areas in Communications*, 14(5):814–823, June 1996.

[5] R. E. Wagner *et al.* MONET: Multiwavelength optical networking. *Journal of Lightwave Technology*, 14(6):1349–1355, June 1996.

[6] *ns.* Network Simulator. http://www.isi.edu/nsnam/ns/.

[7] Diffserv implementation. Nortel Network's Diffserv implementation in *ns.*

[8] O. Gerstel, B. Li, A. McGuire, G. N. Rouskas, K. Sivalingam, and Z. Zhang (Eds.). Special issue on protocols and architectures for next generation optical WDM networks. *IEEE Journal Selected Areas in Communications*, 18(10), October 2000.

[9] Category: Adjustable Fiber Grating Filters. 1999 *Optical Fiber Communication conference proceedings.*

[10] B. Mukherjee. WDM-Based local lightwave networks Part I: Single-hop systems. *IEEE Network Magazine*, pages 12–27, May 1992.

[11] E. Modiano. WDM-Based packet networks. *IEEE Communications Magazine*, pages 130–135, March 1999.

[12] K. Nichols, *et al.* Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. *RFC 2474*, Dec. 1998.

[13] S. Blake, *et al.* An Architecture for Differentiated Services. *RFC 2475*, Dec. 1998.

[14] V. Jacobson, *et al.* An Expedited Forwarding PHB. *RFC 2598*, June 1999.

[15] J. Heinanen, *et al.* Assured Forwarding PHB Group. *RFC 2597*, June 1999.

[16] Kalevi Kilkki. Differentiated Services for the Internet. *Macmillan Technical Publishing*, 1999.

[17] X. Xiao and L. Ni. Internet QoS: A big picture. *IEEE Network*, pages 8–18, March/April 1999.

[18] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, pages: 397–413, 1(4), Aug. 1993.

[19] D. Clark and Wenjia Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, pages: 362–373, 6(4), Aug. 1998.

[20] W. Fang, N. Seddigh and B. Nandy. A Time Sliding Window Three Colour Marker (TSWTCM). RFC 2859, June 2000.

[21] S. Shenker, C. Partridge and R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212, Sept. 1997.

[22] J. Heinanen and R. Guerin. A Single Rate Three Color Marker. RFC 2697, Sept. 1999.

[23] J. Heinanen and R. Guerin. A Two Rate Three Color Marker. RFC 2698, Sept. 1999.

[24] Douglas B. West. Introduction to Graph Theory. Second Edition, Prentice Hall, 2001.

[25] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of computing*, 2(4): 225–231, December 1973.

[26] Teofilo Gonzalez and Sartaj Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23(4): 665–679, October 1976.

[27] V. Sivaraman. TDM schedules for broadcast WDM networks with arbitrary transceiver tuning latencies. Department of Computer Science, NCSU, Raleigh, NC, Master's thesis 1995.

[28] V. Sivaraman and G. N. Rouskas. Packet scheduling in broadcast WDM networks with arbitrary transceiver tuning latencies. *IEEE/ACM Transactions on Networking*, 5(3): 359–370, June 1997.

[29] V. Sivaraman and G. N. Rouskas. HiPeR-l: a high performance reservation protocol with look-ahead for broadcast WDM networks. In *Proceedings of IEEE Computer and Communications*, INFOCOM '97, vol. 3, pages 1270–1277, 1997.

[30] M. Maode, B. Hamidzadeh and M. Hamdi. Efficent Scheduling Algorithms for Real-time service on WDM Optical networks. *Proc. 7th International conference on Computer Communications and Networks*, 1998.

[31] Maode Ma and Mounir Hamdi. Providing Deterministic Quality-of-Service Guarantees on WDM Optical Networks. *IEEE Journal on Selected Areas in Communications*, 18(10), October 2000.

[32] Itamar Elhanny, Jacob Nir, Dan Sadot. A Contention-free packet scheduling scheme for provision of Quality-of-service in Tbit/sec WDM networks. In *Optical Networks magazine*, pages 19–24, July 2000.

[33] Anthony Kam, Kai-Yeung Siu, Richard Bary and Eric Swanson.. Toward best-effort services over WDM networks with fair access and minimum bandwidth guarantee. *IEEE Journal on Selected Areas in Communications*, 16(7), Sept. 1998.

[34] Anothony C. Kam and kai-Yeung Siu. A real-time distributed scheduling algorithm for supporting QoS over WDM networks. In *Proceedings of SPIE*, vol. 3531, Nov. 1998.

[35] Anthony Kam, Kai-Yeung Siu, Richard Bary and Eric Swanson.. A Cell switching WDM broadcast LAN with bandwidth guarantee and fair access. *IEEE Journal of Lightwave technology*, 16(12), Dec. 1998.

[36] Hung-Ying Tyan et al. On supporting time-constrained communications in WDMA-based star-coupled optical networks In *Proc. IEEE 17th Real-Time Systems Symposium*, Dec. 1996.

[37] Bin Wang, Chao-Ju Hou and Ching-Chih Han. On dynamically establishing and terminating Isochronous message streams in WDMA-based local area lightwave networks. In *Proceedings of INFOCOM '97*, pages 1261–1269, IEEE 1997.

[38] Anlu Yan, Aura Ganz and C. M. Krishna. A distributed adaptive protocol providing Real-Time services on WDM-based LAN's. *IEEE Journal of Lightwave technology*, 14(6), June 1996.

[39] S. Selvakennedy et al. Dynamic Scheduling scheme for handling traffic multiplicity in Wavelength division multiplexed optical networks. *Computer Communications and Networks*, pages 344–399, IEEE 1999.

[40] Laura E. Jackson and George N. Rouskas. Optimal scheduling of periodic tasks on multiple identical processors. Dept. of Computer Science, NCSU, Raleigh, NC, Technical Report 1998.

[41] Magnus Jonsson, Klas Borjesson and Magnus Legardt. Dynamic Time-Deterministic traffic in a fiber-optic WDM star network. *Real-Time Systems*, pages 25–33, IEEE, 1997.

[42] M. Jonsson, A. Ahlander, M. Taveniku, and B. Svensson. Time-deterministic WDM star network for massively parallel computing in radar systems. *Proc. Massively Parallel processing using optical interconnections*, pages 85–93, Oct. 1996.

[43] Marco Ajmone Marson et al. All-Optical WDM multi-rings with Differentiated QoS *IEEE Communications magazine*, Feb. 1999.

[44] George N. Rouskas and Mostafa H. Ammar. Analysis and Optimization of transmission schedules for Single-Hop WDM Networks. *IEEE/ACM Transactions on Networking*, 3(2), pages 211–221, April 1995.

[45] F. Jia, B. Mukherjee and J. Iness. Scheduling variable-length messages in a single-hop multichannel local lightwave network. *IEEE/ACM Transactions on Networking*, 3(4), pages 477–487, August 1997.