Time-Slotted Scheduling For Agile All-Photonic Networks

Xiao Liu

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master's of Engineering.

STATEMENT OF CONTRIBUTIONS OF AUTHORS

Three co-authored paper are partially included in this thesis.

In the paper entitled "Performance comparison of OTDM and OBS scheduling for Agile All-Photonic Network", the authors are Xiao Liu, Anton Vinokurov and Lorne Mason. This paper is in the conference proceeding of IFIP Metropolitan Area Network Conference, Vietnam, April 2005. Professor Lorne Mason initially proposed the network architecture and slot-by-slot scheduling concepts. Anton Vinokurov presented the OBS scheduling analysis and assisted in the implementing of OPNET simulation framework. Xiao Liu, the author of this thesis, implemented several scheduling algorithms to perform slot-by-slot scheduling in AAPN and evaluated the efficiency of the algorithms with various network topologies and traffic patterns.

In another paper entitled "A comparison between time slot scheduling approaches for All-Photonic Networks", the authors are Xiao Liu, Nahid Saberi, Mark Coates and Lorne Mason. This paper is in the conference proceeding of IEEE ICICS 2005, Bangkok, Thailand. Professor Mark Coates and Professor Lorne Mason presented the literature review for scheduling in Agile All-Photonic Network, and directed the research on scheduling for AAPN. Nahid Saberi, proposed the frame based scheduling for AAPN, and implemented the algorithm in OPNET simulation framework. Xiao Liu implemented the OPNET simulation framework for frame based scheduling of AAPN, and presented the compared results between frame based and slot based scheduling.

In the paper "Resource Sharing for QoS in Agile All Photonic Networks", Anotn Vinokurov, Xiao Liu and Lorne Mason presented the preliminary results of scheduling for QoS in AAPN. Anton Vinokurov presented the research work on OBS, and Xiao Liu extended the research work on slot-by-slot scheduling to multi-class traffic environment. The research carried out is directed by Professor Lorne Mason.

The supervisor Professor Lorne Mason has attested to the accuracy of this statement.

ABSTRACT

A key issue in the Agile All-Photonic Networks is the centralized scheduling method and the coordination between edge nodes and the core switching node. The focus of this thesis is to propose and evaluate efficient scheduling algorithms for time-slot based AAPN. Assuming some buffering at the edge nodes, we use a two-layer network topology design to simulate the network performance. The scheduling schemes proposed in this thesis are aimed at avoiding contention and improving bandwidth utilization. We employ a simple reservation scheme that guarantees time-slot deliveries. In addition, we implement this time-slot scheduling algorithm in the OPNET simulation environment to evaluate its performance in various scenarios.

SOMMAIRE

Un aspect clé des Réseaux agiles tout-photoniques est la méthode d'ordonnancement centralisé et la coordination entre les noeuds périphériques et le nœud-noyau de commutation. L'objectif de cette thèse est de proposer et d'évaluer des algorithmes d'ordonnancement efficients pour des réseaux agiles tout-photoniques basés la répartition dans le temps. En supposant qu'il existe un tampon à chaque nœud périphérique, nous utilisons une topologie à deux couches pour simuler la performance du réseau. Les algorithmes d'ordonnancement proposes ont l'objectif d'éviter la compétition et d'améliorer l'utilisation de la bande passante. Nous employons un algorithme de réservation simple qui garantie néanmoins les livraisons réparties dans le temps. En plus, nous implémentons cet algorithme d'ordonnancement a répartition dans le temps dans l'environnement de simulation OPNET pour en évaluer sa performance dans maints scénarios.

ACKNOWLEDGMENTS

First, I want to express my gratitude to my supervisor, Professor Lorne G. Mason, for his generous financial support. I am also indebted to him for involving in the AAPN project that gave me so many research and collaboration opportunities. I am especially touched by his patient guidance and helpful advice during the course of this work.

Also other people in the Networking Lab deserve thanks for creating a cosy working atmosphere. In particular I want to thank Anton for many inspiring discussions and guidance in practical issues. It has been a pleasure to work with a person who has such an enthusiastic attitude and provocative ideas.

Finally, I would like to thank my family and my Jiayuan for their endless love and unconditional support during the course of my study.

Contents

Chapter 1

Introduction

As modern networks face increasing capacity demand and diminishing resource availability, network providers are approaching a critical milestone: the optical network. Optical networks, based on the emergence of the optical layer in transport networks, provide higher capacity and reduced costs for new broadband multimedia applications, and advanced digital services.

It is widely accepted that the future Internet will be built on high-capacity, agile optical transport networks. Advances in optical communications have led to high capacities in wavelength-division-multiplexed (WDM) fiber and optical cross connectors. They combine to enable higher flexibility in bandwidth provisioning, simplification of network management, and cost reduction, in particular through the avoidance of optical to electronic to optical conversions.

Agile All-Photonic Network is a recently proposed research network that allows the capability to: (1) perform time-domain multiplexing (TDM), hence dynamically allocating bandwidth to traffic flows as the demand varies, (2) extend the photonic portion of the data path as close to the end-user as possible, (3) quickly transfer very large data blocks, (4) switch large aggregations of traffic, e.g., at line rate 10 Gb/s.

Candidate time domain multiplexing schemes include bit-interleaved TDM, and time-slotted (statistically multiplexed) TDM. The former is similar to WDM in the sense that many small channels are shared by access nodes operating at a peak rate which is a small fraction of the line rate [14]. However, in slotted TDM, one fast channel is shared by access nodes capable of bursting at a rate of 10 Gb/s.

There are two characteristics of this 10 Gb/s slotted TDM networks that significantly impact the architectural design. Firstly, it should be able to operate with long propagation

delays, i.e., there are many data packets in flight at one time in the network—a high-latency environment. Secondly, there is limited processing that can be performed at 10 Gb/s. Many discussions in the literature concern the architecture of very high speed optical multi-access networks in the presence of long propagation delays. There are several difficulties in creating such networks. One difficulty is to design an architecture that provides guaranteed bandwidth. A second problem is to provide efficient and fair bandwidth sharing among users in the presence of both moderately loaded and overloaded network traffic conditions. A third difficulty is to develop algorithms simple enough to execute at the very high rate required by optical networks.

A well-studied architecture for all-photonic network is the overlaid star network. It is an excellent match for the network we are considering because it simplifies the scheduling problem and is proved to be efficient and stable [17]. The overlaid star forms a mesh by connecting each edge node to every other edge node. However, data traversing the network only passes through one photonic switch, resulting in a major simplification of the control problem of ensuring low contention. From the control point of view, each star can be managed independently of the others because there is no data interaction. For each star, resource allocation is concentrated at a single point: the core node in the network.

From the definition of AAPN, no buffering can be used at the core optical switch because we seek a transparent network and delay lines are unattractive. An important issue is how to make fast scheduling decisions that will efficiently use the optical core. Several proposals have been reported on fast scheduling in [12] [15] [16] [4] [7], however, to our knowledge, few of them address the problem in the presence of propagation delay. In the context of high-speed optical router design where line cards corresponding to different input ports introduces large round-trip delay, Minkenberg proposed a stateful protocol in which the scheduler maintains the state of queued traffic [13].

1.1 Thesis Contribution

Contribution of this thesis can be summarized as following:

- Proposed a distributed time-slot scheduling algorithm based on the bipartite matching.
- Adapted the proposed algorithm to schedule differentiated services.
- Implemented the AAPN simulation framework in OPNET Modeler 10.5.

- Evaluated the proposed algorithm in the simulation with respect to the following aspects:
  - Effect of network size
  - Effect of traffic pattern
  - Differentiated service
  - Network scalability: with increased
  - Comparison with other scheduling methods

1.2 Outline of Thesis

The remainder of this thesis comprises 6 chapters. Chapter 2 describes the background of AAPN and time-slotted scheduling systems, and surveys several alternative time-slotted scheduling schemes. Chapter 3 describes the real-time scheduling algorithm that we proposed based on the Parallel Iterative Matching algorithm. In Chapter 4, we provide an overview of the simulation platform implemented in the OPNET Modeler tool. Chapter 5 presents various simulation results and analyses. We conclude with a summary, discussions, and proposals for future work in Chapter 6.

1.3 Published Paper

The content presented in this thesis is partly published in the following three conference articles.

X. Liu, A. Vinokurov, and L. G. Mason, "Performance comparison of OTDM and OBS scheduling for agile all-photonic network," in Proc. IFIP Metropolitan Area Network Conference, Ho Chi Minh City, Vietnam, Apr. 2005. In this paper, we present the preliminary results of scheduling under Local Area Network (LAN) and Wide Area Network (WAN) topologies. Various scheduling approached are discussed and compared in terms of their simulation results in OPNET.

X. Liu, N. Saberi, M. J. Coates and L. G. Mason, "A comparison between time-slot scheduling approaches for All-Photonic Networks", in Proc. IEEE ICICS Conference 2005, Bangkok, Thailand. In this paper, we describe two new OTDM algorithms, one that

performs scheduling on a Slot-by-Slot basis and another that schedules frames of multiple slots. We report and analyze results from OPNET simulations.

A. Vinokurov, X. Liu, L. G. Mason, "Resource Sharing for QoS in Agile All Photonic Networks", accepted by OPNETWORK 2005, Washington D.C., Aug. 2005. Based on the preliminary results from scheduling for QoS in AAPN, we study OTDM scheduling on two DiffServ traffic classes.

Chapter 2

Background

2.1 Agile All Photonic Networks (AAPN)

The Agile All Photonic Networks is a research network proposed to develop an all-photonic network, where the core node will stretch as close as possible to the end-user, and the number of optical-electrical-optical conversions (OEO) in network data paths will be reduced [5]. In contrast to current generation core networks, all-photonic networks have the property that both transmission and switching are performed in the optical domain. Data entering the network is converted to the optical signal, which is not converted back to the electronic domain until the signal reaches the exit-point of the network. The absence of the conversion phase leads to two important advantages: firstly, optical switches have (at least potentially) far greater switching capacity than electronic domain switches, thus removing one of the bottlenecks in a high speed network; secondly, the switches are transparent to data format and bit rate.

2.1.1 AAPN Network Infrastructure

The overlaid star network topology shown in Figure 1.1 is robust to various traffic distributions, as proved in [11]. This topology has a significant influence on the implementation complexity of the bandwidth management mechanisms. The AAPN can be viewed as a distributed switch comprised of edge nodes, where the optical electronic conversion takes place, connected in an overlaid star topology to photonic core crossbar switches employing sub-microsecond photonic switching.

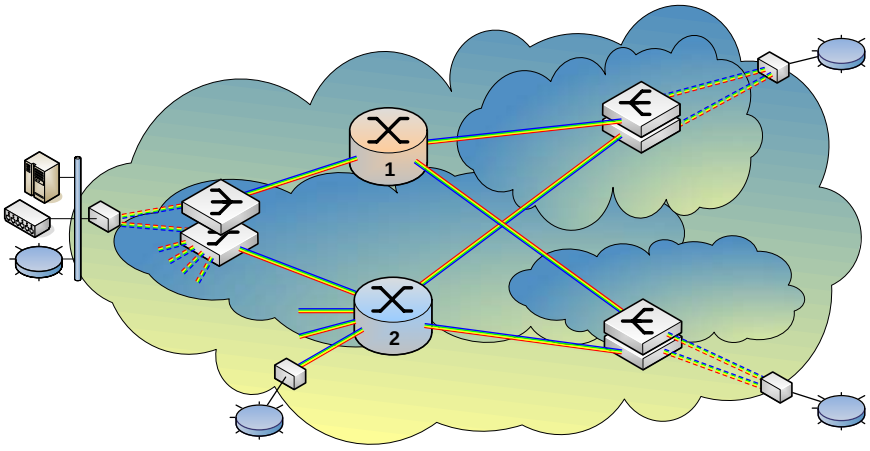


Figure 1.1: Architecture of Agile All Photonic Networks

Synchronization of overlaid stars can be realized by partitioning the virtual output queues in the origination edge nodes according to the core switches serving them. This effectively decouples the synchronization of different core switches and their subtending edge node buffers making the synchronization manageable. The set of VOQs for the different destination edge nodes, located at each ingress edge node, is partitioned into *J* groups, according to the core switch through which the traffic is routed. For the case of deterministic shortest path routing, where only a single path is used for a given origin-destination (O-D) pair, a single VOQ for a specific destination edge node is needed at the origin edge node or ingress. For the more general case of load sharing where traffic is carried on P disjoint paths linking a given O-D pair, then P copies of VOQ buffers are required per edge node for each such O-D pair.

Due to the above decomposition, we consider without loss of generality an optical star network comprised of a non-blocking cross-bar core switch and several edge nodes, which are able to receive and transmit simultaneously. In addition, a source node is attached to each ingress edge node to generate traffic flows for that ingress edge node.

2.1.2 Viable Scheduling Strategy for AAPN: OTDM or OBS?

The overlaid star topology proposed for the AAPN facilitates the introduction of various approaches for time-sharing of link capacity, including optical burst switching (OBS) as well as several optical time division multiplexing (OTDM) techniques. These alternatives differ in terms of the degree of coordination in resource allocation between the edge nodes and the core crossbar photonic switches. For example, optical time division multiplexing (OTDM) requires network synchronization whereas optical burst switching (OBS) does not. OBS can operate in a general class of topologies such as meshes, trees,

rings, stars, etc. However, network synchronization can be realized more easily in tree topologies.

Techniques that have been proposed in the literature include just-in-time signaling architectures for WDM burst-switched networks—JumpStart [20], optical burst switching [19], and Slot-by-Slot routing [9].

Taking a broader perspective, a spectrum of alternatives can be envisaged. Asynchronous Optical Burst Switching minimizes the coordination requirements as it requires neither synchronization nor signaling for time slot reservation. Synchronous slotted OBS version without signaling for slot reservation provides another example where improved traffic handling performance can be anticipated through global synchronization, analogous to that gained in slotted Aloha over pure Aloha systems.

Synchronization which is feasible for overlaid star topologies enables the application of various OTDM techniques, ranging form frame-based deterministic scheduling where edge node transmissions are appropriately clocked to the core switch configuration by means of a round-robin schedule, for example. By allowing for the propagation delay, slots from different endpoints arrive at the core crossbar switch and are switched to their appropriate destinations without output port collisions. Several variations of such deterministic frame-based schemes are feasible by adding additional signaling between the edge and core switches, to make the schedule dependent on the traffic demand.

2.2 Time Slotted Scheduling Schemes

The AAPN architecture consists of one or more stars linking edge nodes where the electronic-optical conversion takes place to the core optical switch. Considering one of these stars, each fiber linking an edge node and the core switch may support an integral number of distinct wavelengths, each carrying signals at a rate of 10 Gb/s. The number of wavelengths required to serve an edge node is determined by the total traffic to and from this edge node to the other edge nodes in the star network. As there is no wavelength conversion in the core, we can model the core switch as one or more space division switch layers, one for each wavelength. The switch configuration states of the core space switch for different wavelength layers are independent. Each wavelength is further time divided into synchronized slots of 10 microseconds each.

For the duration of one time slot, the space switch configuration for each distinct wavelength can be modeled by an assignment in a bipartite graph, consisting of N nodes

representing the input ports of the core switch to N nodes representing the output ports of the space switch. The instantaneous switch configuration is modeled by the assignment which maps input ports to output ports. This is a model of a complete crossbar space switching stage. It assumes that traffic between distinct input ports destined for distinct output ports can be served in parallel on the same wavelength and time slot. If this were not the case, then it would not be a non-blocking core switch. There is another bipartite assignment representing the space switch configuration in the reverse direction for carrying bi-directional traffic. In general, the configurations in the two directions need not be the same if the supported traffic flows are not symmetrical.

Edge nodes that support more than one wavelength will select the wavelength by electronic means before conversion to an optical signal. Once the wavelength is determined the signal propagates in the optical domain without wavelength conversion until it reaches the destination edge switch where it is reconverted back to the electronic domain.

Due to varying sizes of edge node traffics, edge nodes will differ in the number of wavelengths terminated, ranging from a minimum of one, which every edge node must support, up to L wavelengths, depending on edge node traffic demand. Based on demographic data and estimates of traffic generated per terminal edge node, one determines the number of wavelengths required to connect each edge node to the core switch. This will result in a star configuration with an integral link capacity ranging from a minimum of one up to a maximum of L wavelengths for the largest nodes.

Due to the wavelength continuity constraint, communication between nodes supporting multiple wavelengths will employ the higher order wavelengths for communication as well. For example if all N nodes support the first wavelength, while $M<N$ nodes support another one, then the second wavelength cannot be used for communication among the $N-M$ nodes which support only the first one. Communication between the M nodes supporting both wavelengths and the $N-M$ nodes supporting only one wavelength must be done on the first one. This means that the second wavelength is used only for communication among the M larger nodes. For nodes requiring more wavelengths, a similar argument applies as follows.

Communication among type one nodes (i.e., nodes only supporting one color) and type 2 and 3 nodes supporting 2 and 3 wavelengths respectively, must take place on the first wavelength. Communication among type 2 and type 3 nodes take place on the second wavelength, communication among type 2 nodes use the second wavelength and

communication among type 3 nodes use the third wavelength. The approach generalized to higher order nodes up to those supporting all L wavelengths.

A consequence of this assignment is that there will be more time slots within a frame available to nodes with higher order wavelengths as fewer nodes support the higher order wavelengths, where the time slots for each wavelength are of equal duration.

Consider a single wavelength. As the space switch is equivalent to a crossbar, if it can be reconfigured independently for each subsequent time slot, then it will be non-blocking.

However, information required for setting the core switch configuration for a given time-slot must be propagated from the edge nodes to the core switch. Due to the differing propagation times associated with different paths to edge nodes from the core, the calculation of the required configuration cannot be completed until the last request arrives at the core. Moreover, in order to time-division multiplex the different requirements, the launch times must be appropriately determined so that they arrive at the output ports of the core switch without colliding, (i.e., going to the same destination in the same time-slot), and time aligned, (i.e., the traffic arriving at the core switch destined for distinct output ports are synchronized).

One possible way to achieve this proper operation is to use buffering at edge nodes to compensate for the different link propagation delays so as to equalize the arrival times at the core switch. Once these (up to N) signaling requests arrive at the core switch, they are scheduled for transmission in some future time-slot (time necessary includes time to compute switch configuration and signal edge nodes and propagate data from the edge nodes to the core switch). Hence, at least three times the maximum propagation delay associated with the longest link plus the configuration computation time is required from the time of initiating the request at the edge node until it commences service at the core switch.

2.2.1 Slot-by-Slot Scheduling

Slot-by-Slot scheduling is a form of packet switching, with the difference that slots (packets) from distinct sources are scheduled in a coordinated manner. It will have the highest signaling load, but the least delay among the first three alternatives, (1) Slot-by-Slot; (2) frame by frame; and (3) call by call scheduling. The configuration must be recomputed for each slot time. When there are conflicts in the requests (output port collisions), one source is selected for transmission while the others are blocked at the edge nodes until they are granted access by the core. Notice that if a single edge switch buffer is employed

at an edge node and it is served in a FIFO order, this procedure will lead to head of line blocking, and reduce the switch utilization to 0.58 maximum. For a better alternative, the edge node traffic can be separated by placing all traffic destined for the same egress node in FIFO order in the same buffer, leading to N-1 buffers or queues per edge node, one for each egress node. This could be implemented as virtual queues or with physically separate queues. This eliminates the HOL blocking problem.

2.2.2 Frame by Frame Scheduling

Frame-by frame scheduling is another approach to avoid the HOL blocking associated with FIFO Slot-by-Slot operation, where the core switch configuration sequence within a frame, is computed for the entire frame at once. In this alternative, a frame of requests is signaled from the edge switch to the core in each frame cycle. This implies additional delays relative to the Slot-by-Slot approach. However, the throughput can be higher than Slot-by-Slot scheduling because the scheduling is performed once per frame rather than N time per frame for the Slot-by-Slot approach. In this way, frame-by frame scheduling can achieve a global optimum schedule for the frame of traffic rather than N separate optimizations for configuration of the N slot times per frame. The per frame computation of the switch configuration sequence for a frame is more complex than the Slot-by-Slot configuration calculation. The signaling traffic and the configuration calculation are more bursty than that of the Slot-by-Slot counterpart, which is spread out evenly over time.

2.2.3 Call by Call Scheduling

Yet another alternative which further reduces signaling load and computational load for scheduling uses incremental allocations and de-allocations on a call-by-call basis. We dub this mode of operation call by call scheduling. It is similar to the operation of traditional TDM switches such as DMS, with the important difference that the time slot interchange stages are located at the edge nodes while the time division multiplexed space switch stage is located in the core. This option requires signaling for call requests and disconnects. We note that while some traffic in future networks may involve explicit signaling for session initiation, and tear down, there will likely remain connectionless traffic demands, which do not have the notion of session signaling. To handle such connectionless traffic one could inject additional requests for increasing, or decreasing bandwidth between OD pairs to carry the connectionless traffic. This would however, require distinguishing connectionless traffic from connection oriented traffic.

2.2.4 Comparison of Scheduling Schemes

The four schemes outlined above have different signaling requirements, configuration computation loads, and performance as measured by throughput. The options are listed above roughly in order of decreasing signaling load, configuration computation, and throughput capacity. There are differences in the delay and jitter for the various approaches as well.

Chapter 3

Slot-by-Slot Scheduling in AAPN

In the AAPN, one of the main challenges is the control of the traffic going into the core switch and scheduling of the output. Since there is no buffering, we cannot afford to have frequent contention for resources at switches. At a given time, each wavelength on an outgoing link on any photonic switch must be allocated to a single connection through the network. If the core of the network is a mesh of photonic switches, then the coordination of the switches becomes a very challenging problem.

Depicted in Figure 3.1 is the Slot-by-Slot scheduling system, in which slots from distinct sources are scheduled in a coordinated manner.

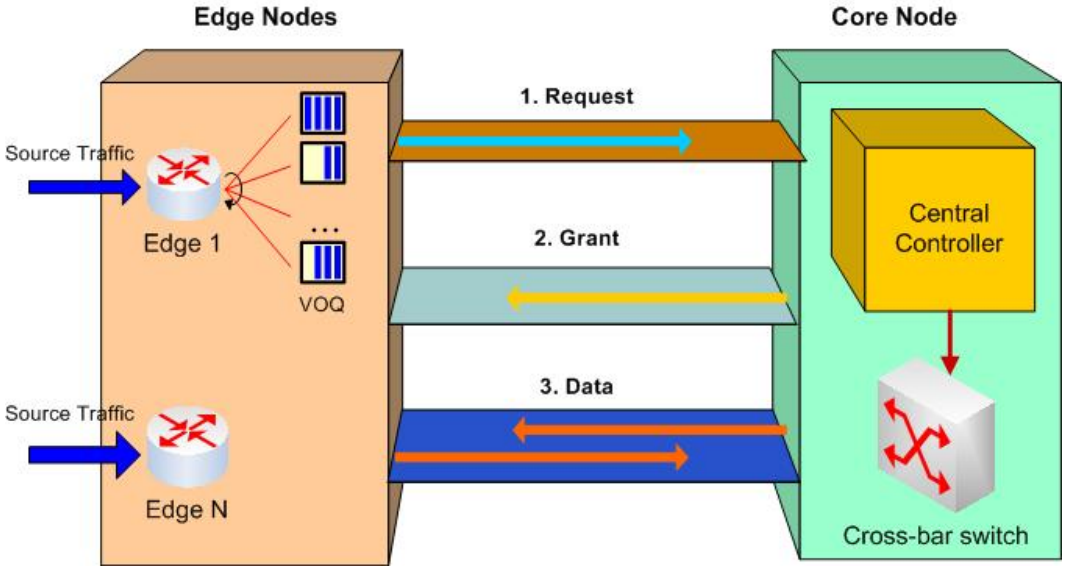


Figure 3.1: Slot-by-Slot scheduling

The configuration of the core switch is computed once for each time-slot, given the matching algorithm is fast enough to find the match within one timeslot. When there are conflicts in the requests, for example, two ingress edge nodes have traffic going to the same egress edge node; one of them is selected for transmission while the others are blocked at the edge nodes until they are granted access by the core.

Because transmission at each edge node is carried out on a time slot basis, a major challenge for global synchronization arises. Since no delay line can be applied in AAPN architecture, we manage the timing by delaying transmission at the edge node side to accommodate the propagation delay between it and the core switch. In addition, to account for the switch reconfiguration time, we apply a narrow guard time that separates slot boundaries. Following is a description of the Slot-by-Slot scheduling method. At each time-slot:

a) Every edge node sends a request to the core switch; the request contains information on whether a specific VOQ has traffic to send or not.
b) A central electronic controller co-located with optical core node sends grants to edge nodes. It will run the matching algorithm to determine which VOQs are to be served in a given future time slot, and configures the cross-bar optical switching matrix at the appropriate future time to transparently switch the arriving traffic slots. After the schedule is computed at the core, grants are sent to those edge nodes that are allocated the future time-slot in question. The grant indicates which VOQ(s) in an edge node will be served and the sending time for edge node to transmit data.
c) The Edge nodes transmit data to core switch via the optical data channel.

In this chapter, we propose a bipartite matching algorithm based scheduling protocol in the Slot-by-Slot system. We will start this chapter by an overview of matching algorithms.

3.1 Scheduling Problem

Most present-day switch architectures use virtual output queuing at the ingress to avoid head-of-line blocking, as shown in Figure 3.1. The key feature differentiating such architectures is whether scheduling of the ingress VOQs is centralized or distributed.

The centralized approach typically uses N input buffers organized by destination (VOQ) combined with an $N \times N$ bufferless crossbar switch core. Within this centralized scheduling category, we can further distinguish among approaches without speed-up

(which are purely input-queued) and those with limited speed-up (which are combined input and output-queued). Both approaches require a centralized matching algorithm to resolve input and output contention. The purely input queued approach requires a bipartite graph matching algorithm, which is described later in this chapter.

3.1.1 Input vs. Output Queuing

One of the central concerns in designing a high speed switching system is limited memory bandwidth [22]. For switches operating at high speed, the speed of the memory, which is a basic building block of queues, becomes a limiting factor. Switching speed is often limited by the rate at which the memory can operate [23]. Under this condition, a switch that makes an efficient use of memory bandwidth can run faster than one that does not. The memory bandwidth requirement varies widely across queuing disciplines [22]. Depending on switch architecture, queuing can take place at different parts of the switch: at the inputs, at the outputs, at both inputs and outputs, or at a centralized location.

The following compares two queuing techniques: output-queuing, centralized shared memory and input-queuing. Output-queuing is referred to as a queueing technique in which all queues are placed at the outputs as shown in Figure 3.2. Switches employing this queuing technique are known as output-queued (OQ) switches. While it is known for its high throughput and its ability to guarantee quality-of-service (QoS), output queuing does not make efficient use of memory bandwidth. Because there are no queues at the inputs, all arriving slots must be immediately delivered to their outputs. Being able to deliver every slot to the output immediately is both desirable and undesirable. In terms of the throughput and the QoS guarantee, it is advantageous because all slots immediately appear at the outputs ready to be considered for transmission, making QoS guarantee possible [24]. A major disadvantage, however, is that simultaneous delivery of all arriving slots to the outputs requires too much internal interconnection bandwidth and memory bandwidth. For an $N \times N$ switch, there can be many slots, one from every input, arriving for any one output simultaneously. To be able to receive all slots at one time, a memory implementing an output queue has to support accesses in a time-slot, as does the interconnection feeding into the memory. This requirement is known as internal speed-up of a switch. An output-queued switch, thus, has an internal speed-up of N [25]. With one read to send one slot to the outgoing link every time-slot, a memory implementing an output queue must operate many times faster than the line rate.

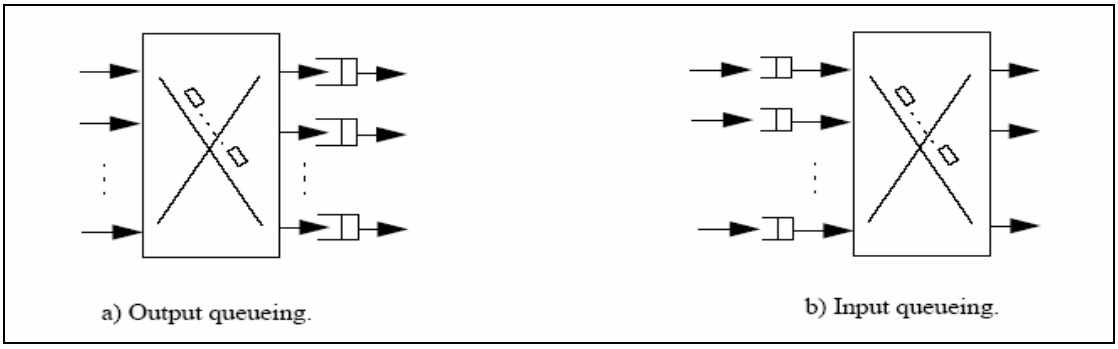


Figure 3.2: Output queuing and Input queueing

Input-queuing, on the other hand, has no speed-up requirement. Queues at the inputs need not be able to receive or send more than one slot simultaneously because at most one slot can arrive at and depart from each input in one time-slot. Thus, the memory is only required to operate at twice the line rate, making input-queuing of interest for high speed switching systems. However, as we know that an input-queued (IQ) switch with a single FIFO queue at each input performs poorly due to head-of-line (HOL) blocking [26]. Nonetheless, this problem can be completely eliminated by a simple queuing technique.

3.1.2 Overcoming Head-of-Line Blocking

Researcher has found a simple technique to completely eliminate HOL blocking [1]. HOL blocking occurs because slots for different outputs share the same FIFO queue. When slots for different outputs share a FIFO queue, a slot that is destined to a free output can be blocked by a slot in front of it that is destined to a different output but has to remain in the queue because its output is busy. As a result, some inputs and outputs are unnecessarily left idle. Karol et al. showed that under certain conditions, HOL blocking results in a $2-\sqrt{2}=58.6\%$ throughput limit [26].

However, one technique called virtual output queueing (VOQ) can completely eliminate HOL blocking. In a VOQ switch, an example of which is shown in Figure 3.1, all inputs maintain a simple queue structure consisting of multiple FIFO queues, one for each output. Now that all slots in each FIFO queue are destined for the same output, no slot can be blocked by a slot in front of it that is destined to a different output; in other words, no HOL blocking can occur. Although VOQ may appear complicated, memory bandwidth required to implement VOQ remains the same as a single FIFO queue because at most one slot can arrive at and depart from each input at a time. Using head and tail

pointers, all queues at an input can share the same physical memory. Logical partitioning of the memory can be either static or dynamic [1].

Efficient memory bandwidth utilization, together with HOL blocking elimination, makes VOQ a viable solution for high-bandwidth switch design. However, the scheduling problem in VOQ switches is more complex than the one in single FIFO switches because VOQ switches maintain more queues at each input than single FIFO switches.

Input maintains FIFOs, one for each output. Time is slotted into time-slots which we refer as slots. Upon arrival, every slot identifies its output destination and thus is queued according to its output destination.

3.1.3 Bipartite Matching for Scheduling

Consider an queued switch connecting *m* inputs to *n* outputs. The arrival process at input *i, $A_i(t)$*, is a discrete-time process of fixed sized packet, we call it a slot. At the beginning of each time slot, either zero or one slot arrive at each input. Each slot contains an identifier that indicates which output *j,* it is destined for. When a slot destined for output *j* arrives at input *i* it is placed in the FIFO queue *Q(i,j)* which has occupancy $L_{i,j}(t)$. *A(t)* is defined as the process of arrivals at input I for output j at rate λ_{ij}, it is considered admissible if no input or out-put is oversubscribed, i.e., $\sum_i \lambda_{ij} \le 1$, $\sum_j \lambda_{ij} \le 1$, otherwise it is inadmissible.

A scheduling algorithm selects a conflict-free match *M* between the set of inputs and outputs such that each input is connected to at most one output and each output is connected to at most one input. At the end of the time slot, if input *i* is connected to output *j*, one slot is sent from *Q(i,j)* to output *j*. Clearly, the departure process from output *j*, $D_j(t)$, at rate μ_j is also a discrete-time process with either zero or one slot departing from each output at the end of each time slot. We define the departure process $D_{i,j}(t)$, rate μ_{ij}, as the process of departures from output *j* that were received from input *i*. To find a matching *M*, the scheduling algorithm solves a bipartite graph matching problem. An example of a bipartite graph is shown in Figure 3.2. In the Request Graph, an edge is a request from *I* to *J*, the meaning of the weights w_{ij} depends on the algorithm. In some algorithms the weight is always equal to one, indicating whether the queue is empty or

non-empty. In other algorithms, the weight $w_{i,j}$ may be integer-valued, indicating the input queue occupancy $L_{i,j}(t)$. In the Matching Graph, the edges connecting input ports to output ports indicating a matching between them.

The scheduling algorithms described in the following parts attempt to match the set of inputs I, of an input-queued switch, to the set of outputs J. We let N be the number of ports. In the matching algorithm, if the queue $Q(i,j)$ is non-empty, $L_{i,j}(t) > 0$ and there is an edge in the graph between input i and output j.

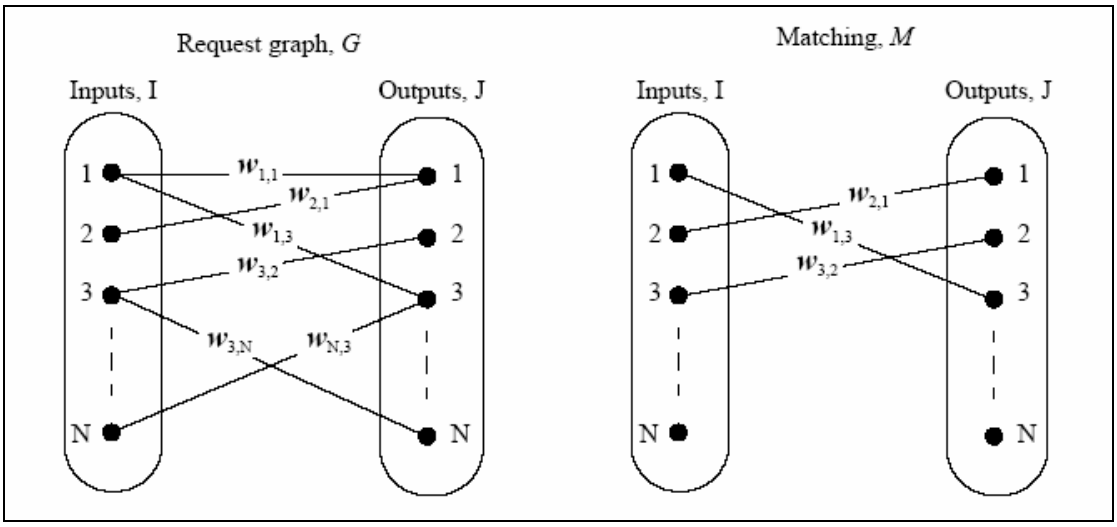


Figure 3.2.: A bipartite matching graph.

There are a number of properties that we desire for all scheduling algorithms [28]:

- Efficiency — an efficient algorithm is one that serves as many input-queues as possible in each match.
- Stability — for a given traffic pattern, we call an algorithm stable if the expected occupancy of every input queue $Q(i,j)$ is finite, i.e., For a given algorithm, we call a stationary traffic pattern sustainable if it does not cause the switch to become unstable.
- No Starvation — we shall describe a non-empty input-queue as starved if, for a given traffic pattern and scheduling algorithm, it remains unserved indefinitely.
- Fast — it is important that the scheduling algorithm does not become the performance bottleneck of the whole system. The algorithm should therefore find a match as quickly as possible.

3.1.4 Bipartite Matching Algorithms

A number of bipartite matching algorithms are commonly used among centralized high-speed scheduling switches. Here we will briefly discuss three of them: Deterministic slot allocation, Parallel Iterative Matching (PIM) and iSLIP.

Deterministic Slot Allocation (DSA)

It is also referred as Round Robin Slot Allocation. The Deterministic slot allocation (DSA) algorithm is an example of how simple it is to achieve 100% throughput for uniform traffic. For an $N \times N$ switch, DSA services each queue deterministically once every time-slots. One possible implementation is as follows: At time-slot *n*, an input is matched to output at *(1+n) mod N* regardless of whether the input has any slot for the output or not.

For each arrival process, each queue can be thought of as an M/D/1 system with a service rate of $1/N$ of the line rate. For uniform traffic, the maximum arrival rate at any queue is always less than $1/N$ of the line rate, and therefore because the service rate is greater than the arrival rate, the system is stable [28].

Parallel Iterative Matching (PIM)

Anderson et al. first introduced an efficient high speed switch scheduling, it was developed for a 16-port, 1-Gbps AN2 switch [1]. It is meant to find a maximal matching within $O(\log N)+3/4$ time. Simulation shows PIM to achieve 100% throughput for uniform traffic.

PIM uses randomness to avoid starvation, and to reduce the number of iterations needed to converge to a maximal matching. PIM attempts to quickly converge on a conflict-free match in multiple iterations, where each iteration consists of three steps. All inputs and outputs are initially unmatched and only those inputs and outputs not matched at the end of each iteration are eligible for matching in the next. The three steps of each iteration operate in parallel on each output and input and are shown in Figure 3.3. The steps are:

Step 1. (Request) Each unmatched input sends a request to every output for which it has a queued slot.

Step 2. (Grant) If an unmatched output receives any requests, it grants to one by randomly selecting a request uniformly over all requests.

Step 3. (Accept) If an input receives a grant, it accepts one by selecting an output among those that granted to this output.

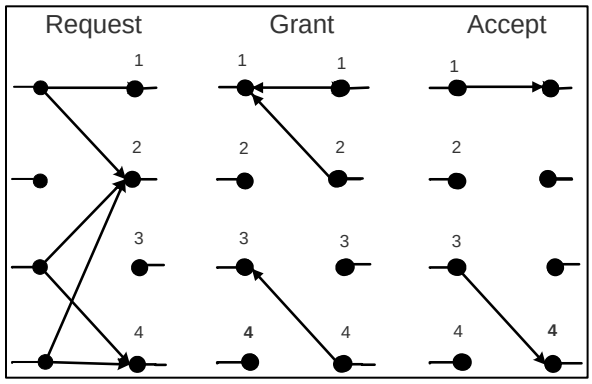


Figure 3.3: PIM matching algorithm

3.2 Motivation for Adapted PIM Algorithm

In contrast to current generation core networks, all-photonic networks have the property that both transmission and switching are performed in the optical domain. Data entering the network is converted into an optical signal, and this signal is not converted back to the electronic domain until the signal has reached the exit-point of the network. The absence of the need for any conversion leads to two important advantages: first, optical switches have (at least potentially) far greater switching capacity than electronic domain switches, removing one of the bottlenecks in a high-speed network; second, the switches are transparent to data format and bit rate. The primary disadvantage is that the tasks necessary for packet switching (buffering, addressing or labeling) are very difficult to implement in the optical domain.

The majority of current core networks have a mesh topology, which is both robust and distributes traffic load over many switches. In the AAPN, the photonic core switches have enormous capacity, so it is appropriate to rethink topology design. One of the main challenges in the AAPN is control of the network; because there is no buffering, we cannot afford to have frequent contention for resources at switches.

3.2.1 Low throughput Due to Large Round-Trip Delay

As we mentioned before, in the PIM algorithm, both input and output sides are selected randomly, without regard to traffic load. This may lead to unfair resource allocation because a VOQ with light traffic load may get access and block another VOQ with heavy traffic load. In each VOQ, packet tracking is applied. We use the queue length

to determine the threshold for sending a request. If the queued packets reach the threshold, a flag is placed in the request, indicating that the edge node has traffic to send. These packets are marked, no more requests will be sent for them in the future. However, under this design, another problem arises. If one request is not granted in central controller, then the packets that made this request will not be served in future. One way to solve this problem could be re-send the request. Since the propagation delay can bring about a long delay, the VOQ would not be able to afford to wait such a long time. For example, if propagation delay between one edge node and the central controller is 50 time-slots, it means that a VOQ should wait at least:

$$\sum_{i=1}^{n} (P_r^i + P_g^{\ i}),$$

Where P_r^i is the propagation delay for the *i-th* request to a packet (from edge node to core node), P_g^i is the grant delay for each request (from core node to edge node). For example, the propagation delay between edge and core node is 50 time-slots, and a packet is granted after request twice, the packet has to stay in VOQ for around 200 time-slots.

3.2.2 Unfairness among Random Selections

Note that in PIM algorithm, the independent output schedulers randomly select a request among contending requests. This has three effects: [1] shows that each iteration will match or eliminate on average at least one of the remaining possible connections and thus the algorithm will converge to a maximal match in iterations. Second, it ensures that all requests will eventually be granted. As a result, no input queue is starved. Third, it means that no memory or state is used to keep track of how recently a connection was made in the past. At the beginning of each slot time, the match begins independently of the matches that were made in previous slot times. Not only does this simplify our understanding of the algorithm, but it also makes analysis of the performance straightforward: there is no time-varying state to consider, except for the occupancy of the VOQs.

But using randomness comes with its problems. First, it is difficult and expensive to implement at high speed: each scheduler must make a random selection among the members of a varying set. Second, the matching decision is made regardless the state of VOQ, this may result a packet blocked in the VOQ, while the packets in other VOQs

which arrive later than it get access. A weighted matching, with respect to the queueing state of each packet, is a good fit to solve this problem.

3.3 Adapted PIM

We have adapted PIM to make it applicable to the AAPN architecture. An edge node does not send a request immediately upon the arrival of a packet in a VOQ. The number of packets in the VOQ for which a request has not been issued must exceed a specified threshold before a new request is sent. Many packets fit in a time slot, so if this policy is not in place, a lightly loaded edge node may request more slots than it needs and be granted a disproportionate number of slots. This can lead to poor utilization within slots and the blocking of heavily loaded edge nodes.Once a request has been issued, the packets associated with that request are marked and no second request is issued for them. This avoids the problem of receiving multiple grants for the same set of packets. We must however ensure that every request is eventually granted, although there may be some time delay in the process. To achieve this, the central controller maintains a list of ungranted requests.

When computing schedules, the controller applies the PIM algorithm, but instead of each output randomly selecting an input in stage one, it selects the input with highest weight request. If multiple requests have the same weight, one of them is selected at random. In the end of scheduling, the list is updated with ungranted requests, and the longer the request stays in the list, the higher the weight it has to be treated. In this way, a request is guaranteed to be granted in the future. Hence, the total delay of a grant to a request is:

$$P_r + P_g + \sum_{i=1}^{n} S_i ,$$

where S_i is the scheduling computation time at the *i-th* try for a request, n is the total number of tries for one request. Let use look again at the case with 50-time-slots propagation, the packets in VOQ are served after 102 time-slots, while it spends 2 time-slots at the central scheduler to be granted. Note that the first propagation delay is decided by the longest propagation delay in whole network because scheduling can not be done until requests from all edge nodes arrive at the central controller. As a practical

matter, unmatched output ports are randomly assigned to a VOQ and a grant is sent despite the absence of a request.

Up to this point, we have explained how the adapted PIM algorithm works in our system, and our proposed algorithm: Consider an $N \times N$ central crossbar switch, where each edge node $i, i \in \{1,...,N\}$, has $N-1$ Virtual Output Queues, corresponding to the each of the other egress edge nodes. The input of the algorithm is status of all VOQs (empty/ nonempty, central controller read the status from request). The output of algorithm is a schedule, which can be interpreted as an s. In any time slot, an ingress edge node can only transmit one packet, and an egress edge node can receive only one packet.

A set for new arriving requests: R = {*(i, j) | Packet want to be sent from edge node i to edge node j*} is kept in the central scheduler and a list for leftover requests from previous time slots or a set *leftover* = {*(i, j) | Leftover request from edge node i to edge node j*}. The schedule for one time slot is determined as follows:

Iteration 1:

•Step 1: Update *R(i , j); i*=0*; j*=0*; I = O = {1,….,N}; Match (i, j) =0.*

•Step 2: Ingress edge node i check *leftover(i, j),* if any, choose egress edge node j with highest entry (weight), put *(i, j)* into *S(i, j)*(a temperate set). If it has all the same entries, choose j randomly; if there is no entry, choose j from *R(i, j)* randomly.

•Step 3: *i*=*i*+1*; if i*<*N*, go to step 3.

•Step 4: Egress edge node j choose an ingress edge node from *S(i , j)*. Fix the matching pair *(i, j)*, remove *(i, j)* from *leftover(i, j)* or *R(i, j)*, put *(i, j)* into *Match(i, j).*

•Step 5: *j*=*j*+1, if j<N, go to step 4.

•Step 6: Put unmatched *R(i, j)* into *leftover(i, j)*. If it already exists in the list, then increase the entry by 1.

Iteration 2 to N:

•Step 1: Check if ingress edge node i has already been matched from previous iteration, if not, choose an egress edge node from *leftover(i, j)*, put into *S(i, j).*

•Step 2: *i*=*i+1; if i*<*N*, go to step 1.

•Step 3: Check if egress edge node j has already been matched from previous iteration, if not, choose an ingress edge node from *S(i, j)*, put into *Match(i, j)*, remove *(i, j)* from *leftover(i, j).*

•Step 4: *j*=*j+1, if j*<*N*, go to step 3.

Further Slot Allocation:
•Step 1: Check the occupancy of each output and input port.
•Step2: Randomly select a empty output and input pair, and matching them.
Update *leftover(i, j)*, and return *Match(i, j)*.

- VOQ places requests if it gets enough new arriving packets and marks these packets.
- Central controller computes schedules for both ungranted requests from previous time-slots and new arriving requests, following the several steps procedure described in PIM algorithm. More iteration can be performed depending on the computation speed of central controller.
- Create grants according to the scheduling, and send them to corresponding edge nodes.
- Update the list for ungranted requests, and assign higher priorities to those requests that stay in the list longer.

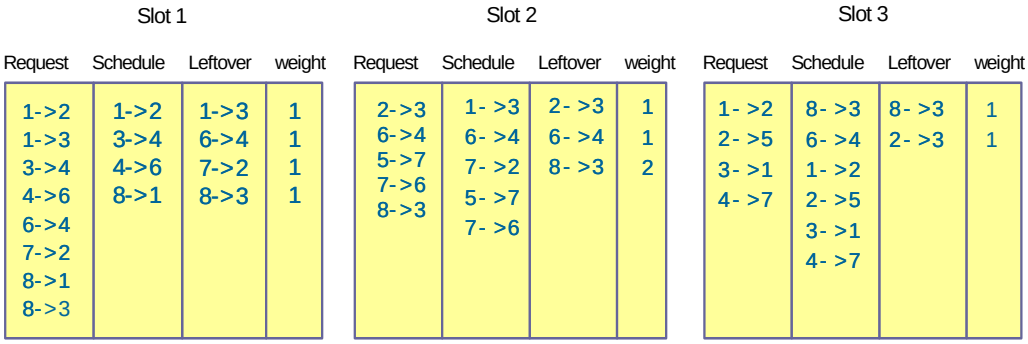


Figure 3.4: Demonstration of request and schedule

3.3.1 Request Monitoring

We have a precise tracking of packets in VOQ of which are requested. In order to prevent sending same requests repeatedly, we created a function "queue_scan." Every time-slot, "queue_scan" goes through every VOQ, and place proper request when there are enough new arrival packets in the queue. Note that a packet will be marked when it has been requested and only a certain number of packets is requested and marked every time-slot, the remaining packets in VOQ will be treated in the next time-slot. If there are not enough unmarked packets in VOQ, request will not be placed, and packets remain unmarked.

3.3.2 Matching Computation

In the central scheduler, we still use PIM as the schedule algorithm; the only change that is made is that those unmatched requests will be added to the next time-slot. For example, at time-slot n, a request from input port 1 to output port 3 is not granted, in time-slot n+1, it will be treated with higher weight over the other newly arriving requests. If another request for the same input and output port arrives in that time-slot, it will be treated in the next time-slot, and so on. In this way, to the edge nodes, there is no ungranted request, but only delayed grant. We now give more explanation about the window size for the request. In order to be fair to every request, we set the request to be for certain amount of packets. Those that have more packets to send will request more and get served more. Besides, it is easier to track the number of packets in VOQs.

Under this design, those requested packets in edge nodes will have a guaranteed service in later time-slots. VOQ buffer loss rate will be decreased. Besides, the packet loss rate could be affected by the number of packets we set for request every time-slot. The packet number bound could be assumed as a window control. Given that a slot size is 100 packets, if we set window size to be 30 packets, the VOQ will be served more frequently, and packet loss rate could be decreased to some extent. However, utilization might be lower, note that a slot is filled by as much packets as possible in the VOQ, and during some time-slot, a VOQ might not have too many packets left, because it just got served in the previous time-slots. On the other hand, if we set window size to 80 packets, utilization could be relatively higher than the previous case, but it might happen that more packets are blocked outside the VOQs.

Figure 3.5 shows the leftover matching over 4 time slots. In time slot one (TS1), input ports A and D both request output ports B, and D is blocked. In the next time slot, TS2, D is selected with higher priority over the other two requests. As a result, the other two requests are blocked and saved for later time slots. The weights associated with request edges are assigned according to how long a request has been staying in the leftover queue.

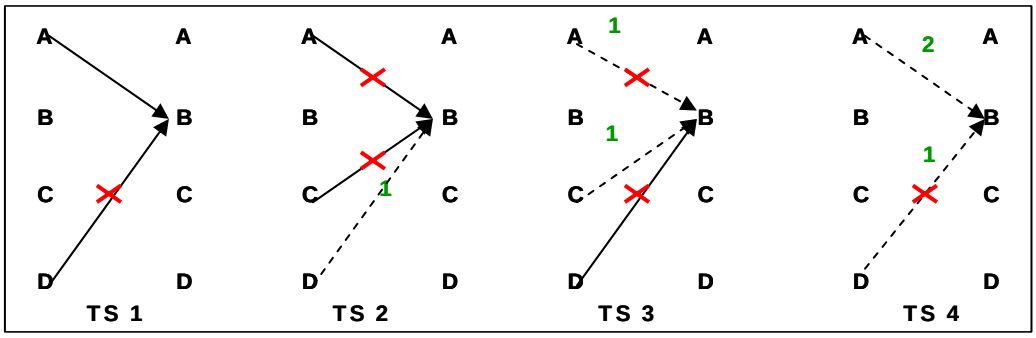


Figure 3.5: Example of weighted matching for one output over 4 time-slots

3.3.3 Further Slot Allocating

PIM algorithm performs matching for multiple iterations. It is proved that matching algorithm converges in log N + ¾ iterations. However, this algorithm only yield a maximal matching result, some input and output port can be left without matching.

In order to increase the throughput of matching algorithm, we add so called "fill up" matching after all iterations are done. At this stage, matching is not done according to the requests made by edge node. It looks through all un-matched input and output port, and assigns matching pair to them randomly. The intuition behind this is that more packets may arrive at the edge node during the round trip time before a grant gets back, and by that time those VOQs, which did not issue any request may have some packets queued already.

Discussion: effect of "fill up" matching could be evaluated with different offered traffic load. Figure 3.6 shows the percentage of "fill up" matching out of total matching over the offered traffic load, and it is gradually decreasing as the offered load grows.

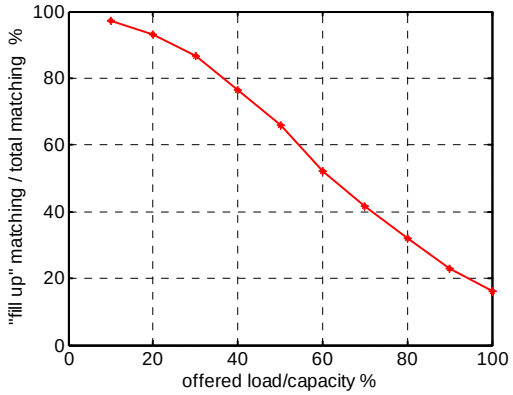


Figure 3.6: ratio of "fill up" matching over total number of matching vs. offered load

This "fill up" matching is possibly able to decrease the delay of request calculating in central scheduler. Since VOQs are visited more often than before, less requests is made, and they are more likely to be matched immediately without being stored in left over list. Under the same simulation conditions, we tracked the "left over matching", which is the matching made for stored left over requests. Result shows that, with "fill up" matching, fewer left over requests is required. Hence, it further reduces the computation of matching algorithm by reducing the new arrival requests as well as left over requests.

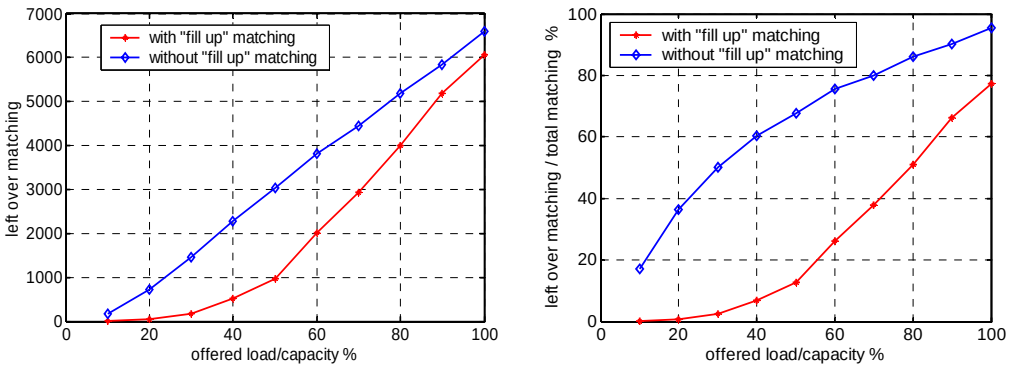


Figure 3.7: effect of "fill up" matching over left over matching

We notice that the influence of "fill up" matching reduced when offered load is high, this is because requests arriving in central scheduler with such intensity that fewer input and output port can be free after matching algorithm. However, "fill up" matching is still useful in relatively decreasing the calculation delay. This comes at the expense of more "jitter" to our system. As an approximate approach, we track the standard deviation of packet buffer delay in each VOQ, in order to check the performance of jitter.

3.4 Discussion on Algorithm Performance

3.4.1 Basic Definition

The goal in slot scheduling is to find a one-to-one match between a nonempty VOQ and a free output. In other words, the scheduler matches an unmatched input to an unmatched output. In this scenario, every unmatched input makes requests to the scheduler telling it which outputs it wants to be matched, and along with every request it can give a weight to indicate its preference. Conceptually, the set of requests can be represented by a bipartite graph, called a request graph, illustrated in Figure 3.2. Associated with every edge is a weight. Consequently, slot scheduling is equivalent to finding a bipartite graph matching [13][67].

Given a request graph, the scheduling algorithm solves a bipartite graph matching problem to find a match graph. To satisfy one-to-one matching, every node can have at most one edge incident as illustrated in the matching graph in Figure 3.2.

Definition 1: An arrival process is said to be admissible if no input and no output is oversubscribed.

Definition 2: The traffic is uniform if all arrival processes have the same arrival rate and all destinations are uniformly distributed over all outputs.

Definition 3: Traffic is called independent and identically distributed (i.i.d.) if and only if:

1. Every arrival is independent of all other arrivals both at the same input and at different inputs.

2. All arrivals at each input are identically distributed.

By considering only the non-empty queues, the scheduler selects a set of conflict-free paths from the set of inputs to the set of outputs. By "conflict-free," we mean that each input in connected to at most one output, and each output is connected to at most one input. The selected queues are then served, which causes them to dequeue their HOL slots and send them along the pre-established paths to the corresponding outputs where the slots can be transmitted on the outgoing link.

We observe that switch scheduling is simply an application of bipartite graph matching: each output must be paired with at most one input that has a slot destined for that output. Basically, there are two interesting kinds of bipartite matches: *maximum* matching and *maximal* matching.

Definition 4: A maximum matching is one that pairs the maximum number of inputs and outputs together. There is no other pairing that matches more inputs and outputs.

Definition 5: A maximal matching is one for which pairings cannot be trivially added. Each node either is matched or has no edge to an unmatched node.

A maximum match must of course be maximal but the reverse is not true. It may be possible to improve a maximal match by deleting some paring and adding others.

3.4.2 Algorithm Complexity

It has been shown that the PIM algorithm finds a maximal matching after $\log_2 N + 3/4$ iterations on average [1]. PIM algorithm is designed to find a maximal match, so that its link utilization can not be as good as maximum match. In the worst scenario, the number of pairings in a maximal match can be as small as 50% of the number of pairings in a maximum match [12]. However, PIM algorithm yields a logarithmic time $O(\log N)$ computation, which is much better than $O(N^2)$ time in maximum matching. And it avoids the starvation of VOQ service. It is said that PIM can find a maximal matching within $\log_2 N + 3/4$ iterations

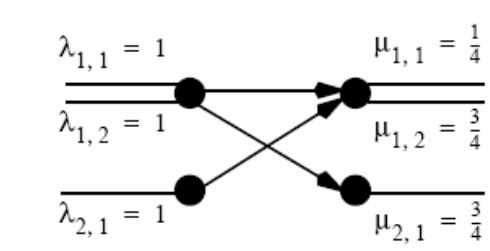


Figure 3.8: Example of unfairness for PIM under heavy inadmissible traffic load

The drawback of PIM comes with its random selection. Studies have shown that PIM could perform miserably under inadmissible traffic [12][15]. An extreme example of unfairness for a 2x2 switch under an inadmissible load is shown in Figure 3.8. We see

examples for which PIM is unfair for admissible but unsustainable traffic. Finally, PIM does not perform well for a single iteration: it limits the throughput to just 63%. This is because the probability that an input will remain ungranted is $(\frac{N-1}{N})^N$, hence as N increases, the throughput tends to $1-1/e \approx 63\%$. Although the algorithm will often converge to a good match after several iterations, the time to converge may affect the rate at which the switch can operate.

3.5 Scheduling for Differentiated Services

3.5.1 Quality of Service in AAPN

Most common carriers, including Telus, Bell Canada, AT&T, and MCI and their associated ISP partners such as Sympatico, are migrating to an IP based converged network infrastructure for provisioning of voice, data, and image services. In addition most cable operators like Videotron, Shaw, Rogers, etc. are adopting an IP based solution for offering a wide range of services to their customers. IP-based services are also supported by wireless LANs for local areas as defined in the IEEE 802.11 series standard example, while WAP, i-mode and 3G wireless technologies can support wireless IP based services over wider areas. To enable a Traffic Engineering, (TE), through source routing, rapid restoration capability and Virtual Private Network (VPN) support, most of the backbone, regional, and local ISPs have already begun or plan to introduce Multi Protocol Label Switching (MPLS) within their IP networks. Initially Sprint indicated it would not migrate to MPLS but rather continue with an over provisioned core network, however they too are now moving in the direction of the other major carriers.

With the relentless growth of existing service traffic demands and the emergence of new services with voracious bandwidth and storage requirements a Quality of Service, (QoS), enabled Internet, it is now widely viewed as the long sought vehicle for service convergence. Many carriers including Telus and AT&T for example have adopted an MPLS/DiffServ architecture for providing Class of Service (CoS) support, which when augmented with an appropriate admission control function can provide the necessary QoS capability for efficient provision of a number of new streaming services, such as packet voice, video on demand and video teleconferencing.

The agile all photonics network can potentially provide a cost/effective high bandwidth/high performance core transport network solution for incumbent and new carriers and ISPs when AAPN comes on line. At that time it is highly likely that among the various client networks that AAPN will serve, including the remaining legacy wire line and

wireless networks, the dominant traffic source will be IP based. Accordingly it is very important to design and position the AAPN to support this dominant traffic source, a significant proportion of which will likely employ the DiffServ/MPLS service architecture and protocols. The incorporation of the MPLS/DiffServ protocols within an AAPN will facilitate *differentiated service provisioning*, cost effective *Virtual Private Network (VPN)* support, and *Traffic Engineering (TE)* in the form of load balancing across the connected overlaid star paths as well as enable *fast restoration capability*.

This partnered research project will focus on the control plane algorithms and signaling protocols resulting from placement of an AAPN within an MPLS enabled networking environment. Benefits accruing from designing the proprietary AAPN architecture and protocols to leverage the MPLS/DiffServ, include Traffic Engineering capability, in the form of load balancing, and fast restoration capability, will be investigated. In particular we will examine alternative methods for restoration including a multi layered approach involving light path as well as restoration of LSPs at the MPLS layer. A second topic to be explored is the issue of providing multiple QoS classes and the resulting impact on service quality, network complexity and cost. We note that the DiffServ/MPLS environment in which an AAPN operates means that this differentiated service capability can be obtained at little additional complexity and has the potential to significantly enhance the utilization efficiency of the AAPN while still providing good service quality to the services that need it. This topic should be of interest to TELUS in view of their rapid role out of a converged IP based MPLS/DiffServ architecture and aggressive strategy for voice migration from the legacy circuit switched network to a packet voice mode. From the AAPN standpoint two basic classes of QoS design options arise along with the associated algorithms and protocols for synchronization, scheduling and signaling. These design options are: Over provisioning with one high quality of service class; Multiple QoS classes. In the latter case two additional questions arise namely, how many QoS classes should be provided and how are they defined and implemented.

3.5.2 Class Based Scheduling

In high speed switching network, the provisioning of Quality-of-Service (QoS) guarantees has become a central topic of research. In this thesis, we focus on provisioning two QoS classes of service: Expedited Forwarding (EF) and Best Effort (BE). EF, also known as premium service, can be used to build a low loss, low delay and low jitter end to end service. The BE class is the same service as that in the current Internet.

The scheduling strategy plays a crucial roll in performing resource allocation and enabling service differentiation. Among the many available scheduling disciplines that can

be implemented in the central controller of the AAPN core switch, the class dependent strict priority (SP) without preemption has been implemented. Based on the Slot-by-Slot scheduling algorithm, we implemented a scheduler for this two-class transport service scheme. The APIM algorithm is extended with the Strict Priority (SP) discipline, in order to allocate time slots to both EF and BE traffics. In order to schedule two classes of traffic, a class based scheduling system is adopted, depicted in Figure 3.9. In coming traffic are associated with different classes, and placed into corresponding class queues.

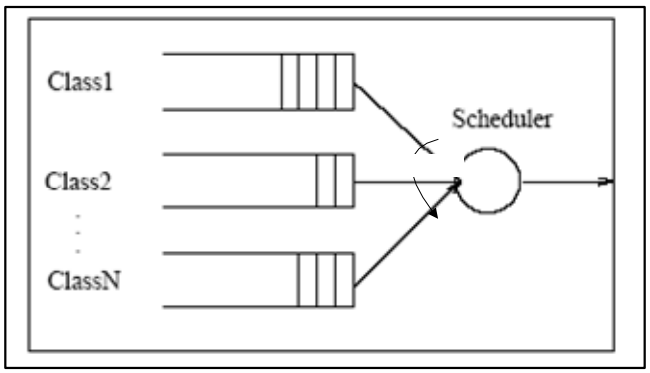


Figure 3.9: Class based scheduling

3.5.3 The Proposed Algorithm

Traffic: We consider two classes of traffics: Expedited Forwarding (EF) traffic and Best Effort (BE) traffic. EF traffic is 20% of total offered load, and the rest is BE traffic.

VOQ: Two VOQs are applied for each destination, storing BE and EF traffic respectively.

Scheduling strategy: At each edge node, VOQs for different destinations are served in round robin fashion. EF traffic has strict priority over BF traffic. During a slot, for the same destination, EF VOQ is always served first, if not enough EF traffic to fill up a whole slot, BE traffic is served then. Two "request" and "scheduling" strategies have been experimented: (1) scheduling is performed merely with EF traffic demand, (2) scheduling is made for both EF traffic and BE traffic, with EF traffic demand has higher priority over BE. Only when no EF request is applied, BE request is considered.

Request

Strategy 1: requests are made based on the traffic demand of EF traffic, BE traffic does not send request at all.

Strategy 2: requests are made upon a combination of BE and EF traffic demand, bounded by different thresholds.

EF and BE traffic for the same destination should have similar behavior. Assuming a Poisson traffic coming to one edge node with arrival rate λ, it is then split into N streams for N destinations. Each stream is still Poisson with arrival rate $\lambda \cdot P_i$, with P_i equals to the probability to i-th destination. Every stream is further split into EF traffic with probability P_e, and BE traffic with probability $1-P_e$. Hence, EF traffic and BE traffic for ith destination are Poisson traffic with arrival rate $\lambda \cdot P_i \cdot P_e$ and $\lambda \cdot P_i \cdot (1-P_e)$.

Schedule

Strategy 1: Adapted PIM algorithm is performed according to the requests from EF traffic. Un-matched EF requests are stored in left-over-list for matching in future slots. A schedule is made for a particular destination, so EF VOQ to this destination is always served first. BE VOQ is only served when EF traffic is not enough to fill up a whole slot.

Strategy 2: In the matching algorithm, left-over EF requests hold highest priority among all the requests. For the new arrival requests, EF requests have higher priority over BE requests. Left-over EF requests are always matched first, if not EF requests exist, check if there are any EF new arrival requests, if not, BE requests are considered. In the end, only un-matched EF requests are stored. Same as strategy 1, schedules are made for destinations; EF VOQs are served first, and then BE VOQs.

Consider an $N \times N$ central crossbar switch, where each edge node $i, i \in \{1,...,N\}$, has 2(*N*-1) Virtual Output Queues in total for both traffic classes. In each time slot, packets can be transmitted from any chosen VOQ within both traffic classes. The input of the algorithm is status of all VOQs (empty/ nonempty, central controller read the status from request). The output of algorithm is a schedule, which can be interpreted as an s. Two lists for left-over requests from previous time slots are kept for both classes: *Leftover_ef* = {*(i, j) | EF packet want to be sent from edge node i to edge node j*}, and *leftover_be* = {*(i, j) | BE packet want to sent from edge node i to edge node j*}, is kept in the central scheduler. A list for new arriving requests R_{BE} = {*(i, j)|BE packet want to be sent from edge node i to edge node j*}, and R_{EF} = {*(i, j)|EF packet want to be sent from edge node i to edge node j*} is the input to scheduling algorithm. The schedule for one time slot is presented as follows:

Iteration 1:

•Step 1: Update *R(i , j); i=0; j=0; I = O = {1,….,N}; Match (i, j) =0.*

•Step 2: Ingress edge node i check $leftover_{EF}(i, j)$, if any, choose egress edge node j with highest entry (priority), put *(i, j)* into *S(i, j)* (a temperate set). If there is no request from I in $leftover_{EF}(i, j)$, then check $leftover_{BE}(i, j)$. If it has all the same entries, choose *j* randomly; if there is no entry, choose j from $R_{EF}(i, j)$ randomly.

•Step 3: *i=i+1; if i<N*, go to step 3.

•Step 4: Egress edge node j choose an ingress edge node from *S(i , j)*. Fix the matching pair *(i, j)*, remove *(i, j)* from the further matching.

•Step 5: *j=j+1, if j<N*, go to step 4.

•Step 6: Place unmatched *requests* into leftover list for its traffic type. If it already exists in the list, then increase the entry by 1.

Iteration 2 to N:

•Step 1: Check if ingress edge node *i* has already been matched from previous iteration, if not, do as step 2 in iteration 1, without checking leftover queues.

•Step 2: *i=i+1; if i<N*, go to step 1.

•Step 3: Check if egress edge node j has already been matched from previous iteration, if not, choose an ingress edge node from *S(i, j)*, put into *Match(i, j)*, remove *(i,* j) from R_{EF} *(i, j)* or $R_{BE}(i, j)$.

•Step 4: *j=j+1, if j<N*, go to step 3.

Further Slot Allocation:

•Step 1: Check the occupancy of each output and input port

•Step 2: Randomly select a empty output and input pair, and matching them

Update $leftover_{BE}(i, j)$ and $leftover_{EF}(i, j)$.

Return *Match(i, j).*

Chapter 4

Experimental Modeling in OPNET

For verification and comparison purpose, we implemented these scheduling approaches in the simulation framework of OPNET Modeler 10.5 discrete-event simulator software.

4.1 OPNET Simulator

OPNET (Optimized Network Engineering Tool) provides a comprehensive development environment for the specification, simulation and performance analysis of communication networks. A large range of communication systems from a single LAN to global satellite networks can be supported. Discrete event simulations are used as the means of analyzing system performance and their behavior.

4.1.1 The Key Features of OPNET

Modeling and Simulation Cycle

OPNET provides powerful tools to assist user to go through three out of the five phases in a design circle (i.e. the building of models, the execution of a simulation and the analysis of the output data).

Hierarchical Modeling

OPNET employs a hierarchical structure to modeling. Each level of the hierarchy describes different aspects of the complete model being simulated.

Specialized in Communication Networks

Detailed library models provides support for existing protocols and allow researchers and developers to either modify these existing models or develop new models of their own.

Automatic Simulation Generation

OPNET models can be compiled into executable code. And executable discrete-even simulation can be debugged or simply executed, resulting in output data.

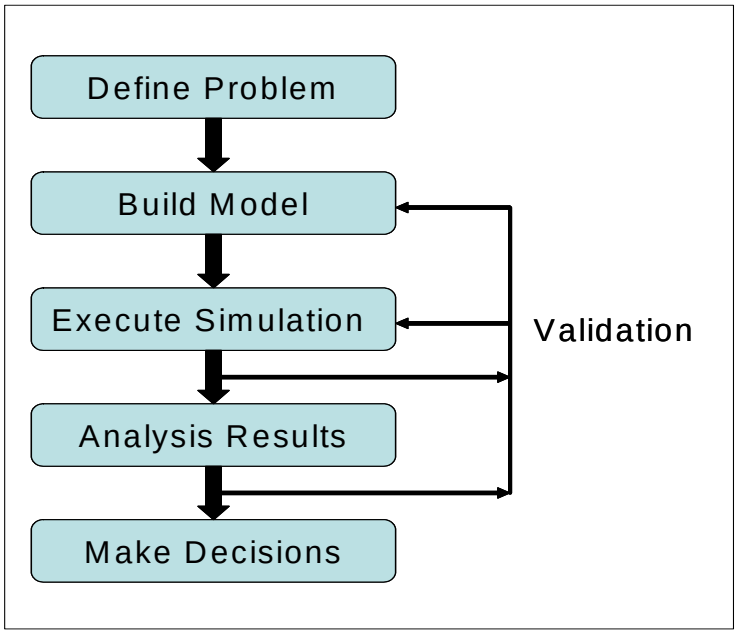


Figure 4.1: Flowchart for building OPNET Model

This sophisticated package comes complete with a range of tools which allows developers specify models in great detail. Identify the elements of the model of interest, execute the simulation and analyze the generated output data:

- Hierarchical Model Building
  - ✧ Network Editor – deploy network topology models
  - ✧ Node Editor – define functionalities of network objects
  - ✧ Process Editor – control underlying function of modules
- Running Simulations
  - ✧ Simulation Tool – define and run simulation
  - ✧ Debugging Tool – interact with running simulations
- Analyzing Results
  - ✧ Probe Editor – collect data results

- ✧ Analysis Tool – collect statistical results
- ✧ Filter Tool – data processing
- ✧ Animation Viewer – dynamic behavior

4.1.2 Hierarchical Modeling

OPNET provides four tools called editors to develop a representation of a system being modeled. These editors, the Network, Node, Process and Parameter Editors, are organized in a hierarchical fashion, which supports the concept of model lever reuse. Models developed at one layer can be used by another model at a higher layer. We will briefly introduce each of the modeling domains. The Parameter Editor is always seen as a utility editor, and not considered a modeling domain.

Network Model

Network Editor is used to specify the physical topology of a communication network, which defines the position and interconnection of communicating entities, i.e., node and links. The specific capabilities of each node are realized in the underlying model. A set of parameters or attributes is attached with each model that can be set to customize the node's behavior. A node can either be fixed, mobile or satellite. Simplex or duplex point-to-point links connects pairs of nodes. A bus link provides a broadcast medium for an arbitrary number of attached devices. Mobile communication is supported by radio links. Links can also be customized to simulate the actual communication channel.

The complexity of a network model would be unmanageable where numerous networks were being modeled as part of a single system. This complexity is eliminated by an abstraction known as a subnetwork. A subnetwork may contain many subnetworks. At the lowest level, a subnetwork is composed only of nodes and links. Communications links facilitate communication between subnetworks.

Node Model

Communication devices created and interconnected at the network level need to be specified in the node domain using the Node Editor. Node models are expressed as interconnected modules. These modules can be grouped into two distinct categories. The first set is modules that have predefined characteristics and a set of built-in parameters. Examples are packet generators, point-to-point transmitters and radio receivers. The second group contains highly programmable modules. These modules referred to as processors and queues, rely on process model specifications.

Each node is described by block structured data flow diagram. Each programmable block in a Node Model has its functionality defined by a Process Model. Modules are interconnected by either packet streams or statistic wires. Packets are transferred between modules using packet streams. Statistic wires could be used to convey numeric signals.

Process Model

Process models, created using the process editor, are used to describe the logic flow and behavior of processor and queue modules. Communication between process is supported by interrupts. Process models are expressed in a language called Proto-C, which consists of state transition diagrams (STDs), a library of kernel procedures, and the standard C programming language. The OPNET Process Editor uses a powerful state-transition diagram approach to support specification of any type of protocol, resource, application, algorithm and queuing policy. States and transitions graphically define the progression of a process in response to events. Within each state, general logic can be specified using a library of predefined functions and even the full flexibility of the C language. Process may create new processes (child process) to perform sub-tasks and thus is called the parent process.

4.2 Simulation Framework

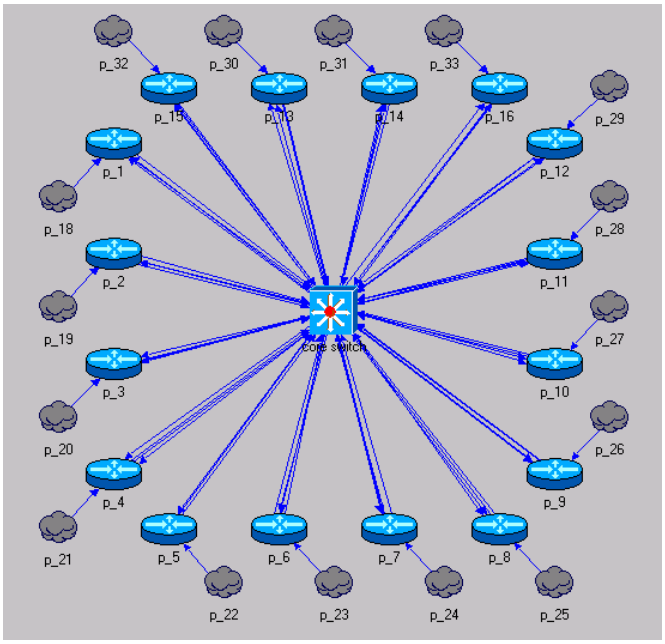


Figure 4.2:Implemented network framework in the node editor

Since we are interested in payload protocol-independent algorithm performance, we decided to simplify our model by implementing all building blocks in form of single Modeler "node" entity, constructing edge and core node models as "process models" within that node. That approach exempts us from necessity programming links models, transmitters, packet encapsulation and routing.

An example of simulation framework is shown in figure 5.2. Propagation delays were generated randomly for both the Local Area Network (average distance is 300 km) and the Wide Area Network (average distance is 1000 km.).

A single node is implemented in the Network Model, representing the whole AAPN network. Several separated processor in the Node Models are deployed to function as the objects in Network. Then we further define the functionalities of these objects in the Process Model. The network model consists of following objects:

- **Traffic sources** are representing the legacy part of network sending data to the AAPN at rates up to 10 Gbit/s. They generate data packets with specified inter-arrival time (Poisson process) and variable size (exponential distribution). Destination of the packet is chosen by a random process (uniform and weighted uniform distribution).
- **Edge node** takes incoming data stream from traffic source and sends packets to network. Edge node implements the client part of scheduling process.
- **Packet streams** providing packet delivery from edge node to core node and back, with a given propagation delay.
- **Core node**, which implements server portion of scheduling process as well as switched incoming packets according to computed schedule.

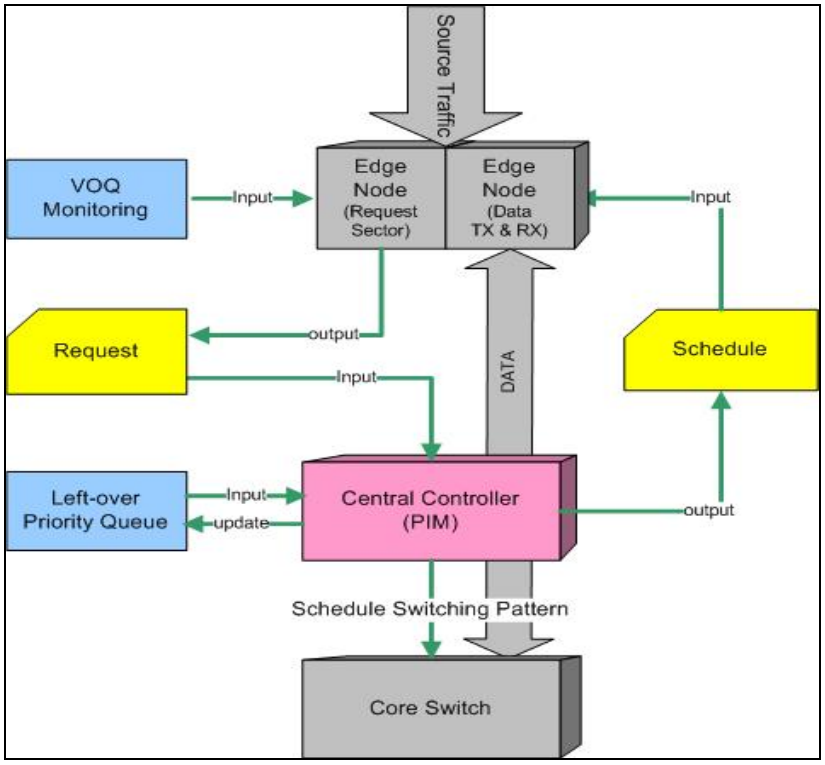


Figure 4.3: Logic diagram for slot based scheduling system

4.2.1 Modeling Slotted Behavior

The discrete simulation property of OPNET Modeler facilitates us in creating the slotted behavior of system model. As we mentioned in previous part, the OPNET is an event driven simulator, while in the simulation environment, the events are referred as interrupts. When a process is invoked by an interrupt, it usually is in a state in which it expects a limited set of interrupts. The first concern of the process will be to determine the type of the incoming interrupt, so that it can tailor subsequent processing appropriately.

There are several kinds of interrupts specified in OPNET Modeler: Self, remote, or multicast interrupts. To distinguish the purpose of such interrupts, and hence provide the receiving process with context-sensitive processing ability, an integer code is associated with self, remote, and multicast interrupts. In order to realize the slotted behavior, every single process nested in the Node Model is scheduled a self-interrupt with the same time intervals (time-slot). The self-interrupt is repeatedly scheduled with an offset time added to the current simulation time. When a self-interrupt is detected, corresponding block of

codes are executed. For instance, in Figure 4.4, when a self-interrupt for starting a time slot occurs, the "request" state in the FSM is visited to generate and send request to core node.

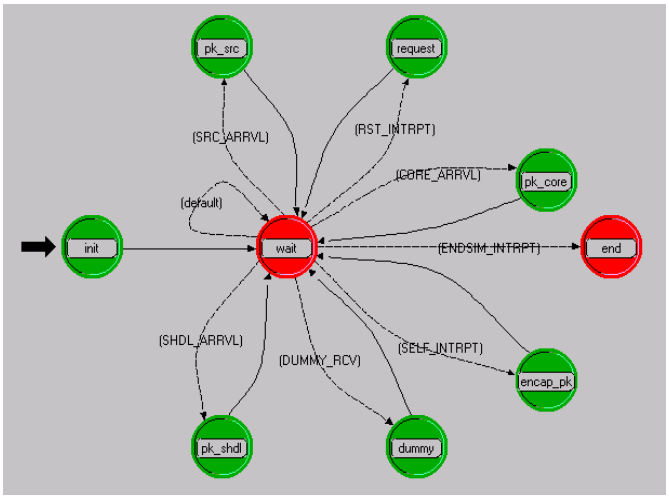


Figure 4.4: Edge node process model

4.2.2 Edge Node Model

An edge node has a number of Virtual Output Queues (VOQ) by number of destination edge nodes. The VOQ size is fixed, in case of queue overflow due to service starvation, newly arrived packets from source are dropped and the queue loss counter is increased.In case of output port collision in the core node (burst mode), data is also lost, and the corresponding counters are also updated.

In the Figure 4.3 shown above, the edge node is divided into two sectors: request sector and data transmit and receive sector. Request sector is in charge of making request decisions. VOQ monitoring is employed in this sector to count the new arrival packet, and mark the packet that have already requested.

In the data transmit and receive sector, schedules from the central scheduler are first stored in a scheduler list. In the beginning of each time-slot, the scheduler information is extracted from the first entry of this list, and deleted after data from corresponding VOQ is scheduled for the transmitted.

Figure 4.4 shows the process model of edge node. The FSM depicted here can be viewed as a Markov Chain. The states in green represent the recurrent states with period

defined in the interrupt function. The conditions on the transition connectors indicate the interrupt associate to the transition. The functionality of some states is listed as following:

- "init" state: the simulation begins with this state, collecting initial simulation parameters, identifying network topology, and obtain simulation input values (e.g., time-slot length, bandwidth).

- "wait" state: dummy state between each transition to another state. Upon receiving an interrupt, it transits to the corresponding state.

- "request" state: is associated with the request decision making sector, VOQ monitoring is implemented in this part, and it returns a request packet after execution.

- "end" state: upon receiving the end of simulation interrupt, simulation terminates, and enters this states, collects the results, and outputs to a file.

4.2.3 Source Node Model

As shown in figure 4.2, a source node is attached to each edge node, modeling traffic generated from the legacy network. We build the source model capable to model traffic patterns such as:

— Poisson arrival traffic: with arrival rate $\lambda\ pk/\mathrm{s}$, packet size is exponential distributed with the mean value $\bar{s}\ bit/pk$. Finally, the traffic offered load to each edge node is calculated as: $\lambda \cdot \bar{s}\ bit/s$.

Under Poisson arriving traffic, the distribution of traffic destination is also specified in the source node model. As defined in Chapter 3, with the uniform traffic all destinations are uniformly distributed over all egress edge nodes. Besides, to study the performance of our scheduling algorithm under non-uniform traffic, we adopt the traffic model, which is characterized as follow:

$$\lambda_{ij} = \begin{cases} 0 & if\ \ i = j \\ \lambda(w + \dfrac{1-w}{N-1}) & if\ \ j = (i+1) \bmod N \\ \lambda \dfrac{1-w}{N-1} & otherwise \end{cases},$$

Where λ_{ij} represents the traffic intensity from input *i* to output *j*; *w* is the non-uniform coefficient, value changes from 0~1; N is the number of edge nodes. Note that the offered load for each ingress edge node and egress edge equals to 1.

Under this traffic model, some of the edge nodes could receive more traffic from one of the other edge nodes. Note that the offered load to every ingress edge node and egress edge node is:

$$\lambda_i = \sum_{j=0}^{N-1} \lambda_{ij} = \lambda[w + (N-1)\frac{1-w}{N-1}] = \lambda = \sum_{i=0}^{N-1} \lambda_{ij} = \lambda_j,$$

which generates admissible traffic to every input and output pair.

4.2.4 Core Node Model

In the AAPN scheduling system, there is central scheduler co-located with the optical switch. Central scheduler is in charge of controlling incoming traffic from edge nodes and scheduling switching patterns of optical switch. We integrate them into one process model. The scheduling algorithms are implemented in function blocks which are transparent to the OPNET simulation. This facilitates us to program different algorithm in C/C++, and load them independently into OPNET simulation environment. The main states in core node model are:

- "init" state: identify network, initialize simulation parameters.
- "request_arrival" state: upon receiving an interrupt from packet stream, obtain request packets from the corresponding packet streams, check request information, and convert them to the input to the bipartite scheduling algorithm block.

- "scheduling" state: upon receiving a self-scheduled interrupt for a new time-slot, invoke the scheduling algorithm function block, and output the schedule after calculation. Encapsulate the scheduling information into separated packets, and send them to corresponding edge node through assignment packet stream.

4.2.5 Implementing Frame Based Scheduling

Frame by frame and Slot-by-Slot schemes were implemented in a common way and share the same set of simulation parameters. We collect statistics to determine the performance characteristics, in particular packet loss and utilization of bandwidth, and end to end delay.

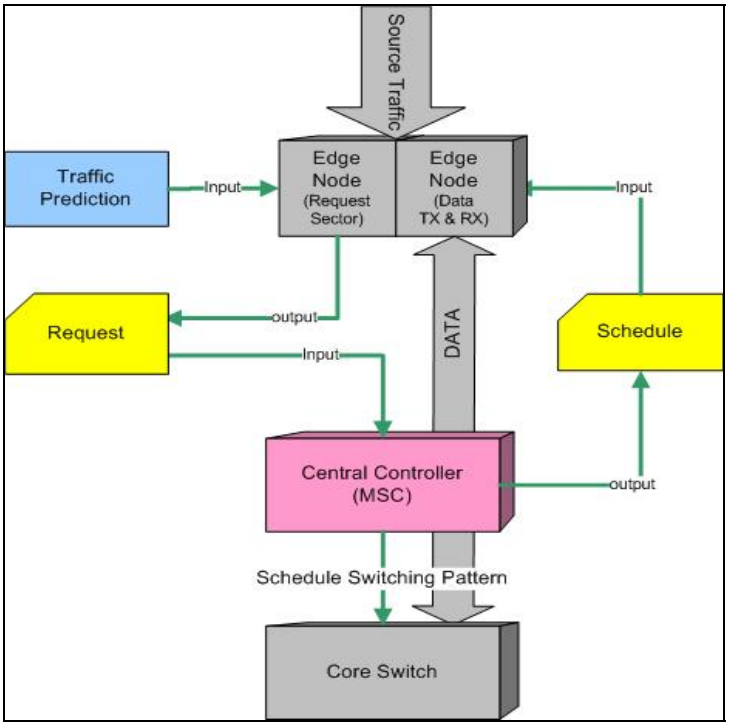


Figure 4.5: Frame based scheduling system for Frame

4.3 Collected Performance Statistics

We collect statistics to determine the performance characteristics, in particular packet loss and utilization of bandwidth, and end to end delay. Initial information is collected by Edge node process models and stored at the end of every simulation at the end of output

log file. Customized Matlab software is then used to extract and display various 2D and 3D dependencies (i.e., packet loss vs offered load vs queue length). For each value of offered load (amount of traffic generated by the source to the network) a discrete-event simulation is performed. To collect reliable statistics more than 1 million packets were forwarded by each edge.

The number of edge nodes considered ranges from 2 to 32. Propagation delays were generated randomly for both the Metro Network (average distance is 10 km) and the Wide Area Network (average distance is 1000 km.). Optical Burst Switching model and various Optical Time Division Multiplexing models (discussed below) were implemented in a common way and share the same set of simulation parameters. Following are the default parameters used in the simulations:

- Time slot (τ): 10 μsec = 10-9 sec.
- Request bound: 70~80 packets.
- Bandwidth of link (capacity): 10 Gbps.
- Arrival rate of uniform Poisson traffic: λ = 0 to 10,000,000 packets/sec.
- Mean packet size: 1000 bits.
- Slot size: we define the slot size by multiplying the line rate of link and the duration of one time-slot, which is: $10^{-5} \times 10^{10} = 10^{5} bits$.
- Topology: Metropolitan Area Network (MAN) and Wide Area Network (WAN).

We now define the statistics we evaluate:

Packet End to End Delay = arrival time at ingress node - arrival time at egress node

$$\text{Packet Loss} = \frac{\text{Packets generated at the traffic source - Packets received at the destination}}{\text{Packets generated at the source}},$$

$$\text{Utilization} = \frac{\text{Aggregate bits of data sent from edge node to core}}{\text{Capacity} \times \text{Simulation time}},$$

$$\text{Offered load} = \text{Packet arrival rate } (\lambda) \times \text{Packet size}.$$

4.3.1 Packet End to End Delay

The end-to-end delay of each packet through a switched network is the sum of the delays it experiences passing through all the switch and route. To determine the end to end delay a packet experiences in the network, four components must be considered:

- Buffer delay is the time spent by the packet in the server queue (buffer) while waiting for transmission. This delay is most difficult to bound. Note that in AAPN system, queuing delay only happens at the edge node.

- Transmission delay is the time interval between the beginning of transmission of the first bit and the end of transmission of the last bit of the packet on the output link. This time depends on the packet length and the rat of the output link.

- Propagation delay is the time required for a bit to go from the sending switch to the receiving switch (or host). This time depends on the distance between the source host and the destination host. It is independent of the scheduling scheme. In our simulation, the largest distance between edge node and core node measured by light speed.

- Processing delay is any packet delay resulting from processing overhead that is not concurrent with an interval of time when the server is transmitting packets.

Note that, in our system, Transmission delay at the edge node is bounded by the line rate of optical link at 10 Gb/s. Propagation delay is determined by the distance between edge node and central switch, it can be very crucial in a large scale network, such as Wide Area Network and Regional Network. In most existing work on the scheduling disciplines, the propagation delay is ignored when come to analyze the algorithm performance.

Chapter 5

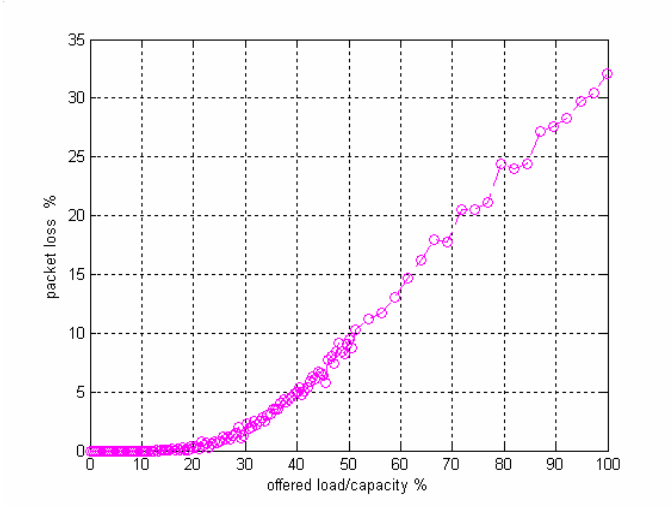
Data Analysis

The scheduling algorithm that we proposed in Chapter 3 is studied in various simulation scenarios. With different topologies and traffic patterns, we explore the algorithm's capability of improving bandwidth utilization, while keeping packet loss rate relative low. Performance results obtained by simulation show that the scheduling algorithm performs well in terms of packet loss and bandwidth utilization, while keeping packet delay sufficiently low to meet real time QoS requirements. Slot-by-Slot scheduling is suitable for MANs, while frame based scheduling schemes with signalling is appropriate for either WAN or MAN applications.

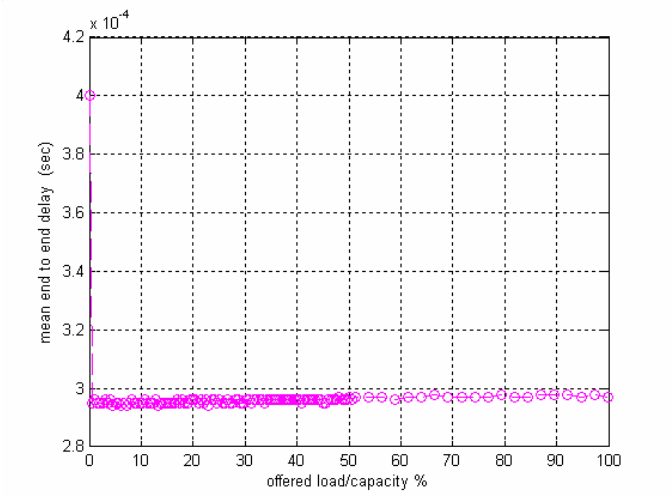
5.1 Metropolitan Area Network (MAN)

5.1.1 Static Switching State

With small networks, for instance, 4-edge-node networks, we can use *static switching state* scheduling, which assigns one switching-state out of nine every time slot, without knowing the traffic information at each edge node. The states are chosen uniformly at random. Note that only nine switching states exist when it is a 4×4 cross-bar switch. Simulations for this case are carried out in a Metropolitan Area Network (MAN) topology, with the VOQ buffer size set to be 100 packets.



(a)

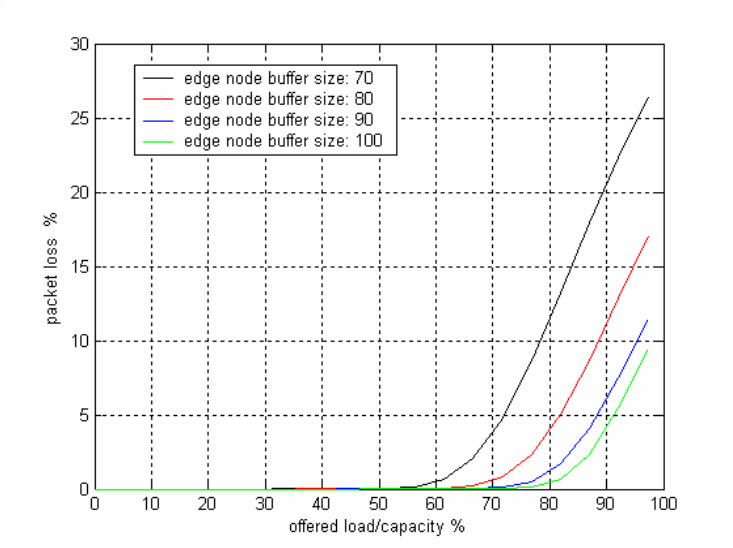


(b)

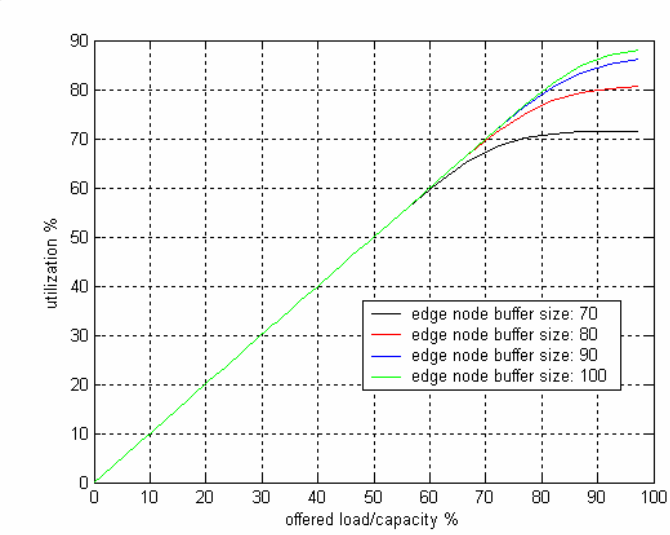
Figure 5.1: (a): packet loss rate vs. offered load; (b): mean packet end to end delay

However, when the network grows in size, we can not use static switching states. For example, when we have 8 edge nodes, there are 8! switching states. A certain fair scheduling algorithm must be implemented. A deterministic slot allocation scheduling algorithm is studied in the following subsection.

5.1.2 Deterministic Slot Allocation



(a)



(b)

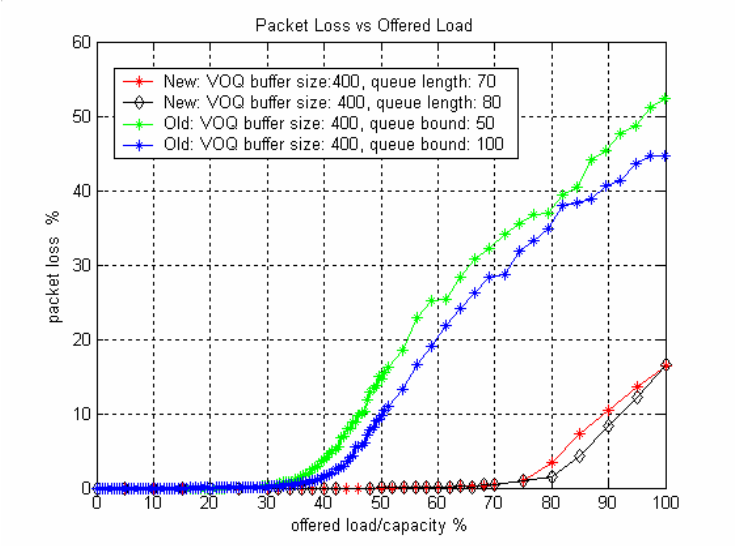
Figure 5.2: Deterministic slot allocation under MAN topology (8 edge nodes).
(a): packet loss vs. offered load; (b): utilization vs. offered load

Figure 5.2 shows the performance of round robin allocation for MAN topology. Utilization of bandwidth can reach 88% and packet loss is kept below 10%. This is because every VOQ is served on a regular basis, and VOQ overflow is less likely. For an 8-edge network, every VOQ buffer is served every seven time-slots. Hence, the probability of overflow a buffer is essentially low. For an 8-edge-node network, we can use a Round Robin Scheduling in the central scheduler. Every ingress edge node will have traffic go to one of the other seven egress edge node periodically. In another word, every VOQ buffer will be served every seven time-slots.

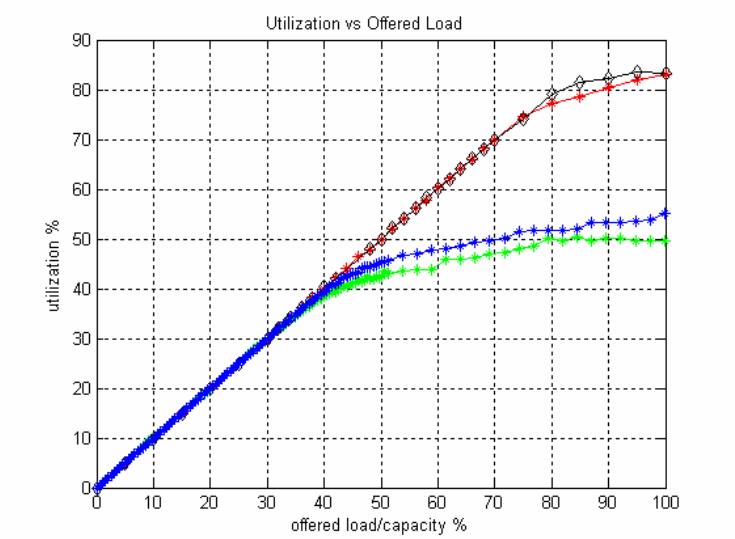
Consider the case where the arrival rate is $\lambda = 5 \times 10^6 \ pk/\mathrm{s}$. One VOQ is going to be served in every seven time-slots. During this time, the number of new arriving packets is $(7 \times 10^{-5} \sec) \times (5 \times 10^6 \ packets/\sec)/7 = 50 \ packets$. Meanwhile, the slot size is 100,000*bits*, which is comparable to 100 *packets*. This results in 50% utilization. Hence, in ideal conditions, i.e., when the offered load equals the capacity, a 100-packet-buffer will give around 100% utilization. From the simulation results, when buffer size grows bigger than 100 packets, utilization can achieve 90%.

5.1.3 PIM and Adapted PIM algorithm

In Figure 5.3, a significant performance improvement is shown for Adapted PIM algorithm. It is useful to point out that the queue bound and queue length both indicate the threshold for sending a request from a certain VOQ, with the difference that PIM counts all the packets currently stored, while Adapted PIM counts the accumulated number of packets since last request. As discussed in Chapter 3, without tracking the update of VOQ state, more than one request can be made for the same packets, which may result in low utilization of allocated time slots. This is the reason why PIM performs poorly in our system. With Adapted PIM, a one-to-one correspondence exists between each request and packets, so that no excess request is made for the same packets.



(a)



(b)

Figure 5.3: Performance comparison of PIM and Adapted PIM
(a): packet loss vs. offered load; (b): utilization vs. offered load (with "new" indicate Adapted PIM, and "old" as PIM)

Regarding the packet loss rate of Adapted PIM, we observe similar performance as DSA. Particularly, Figure 5.3 shows that Adapted PIM requires larger buffer size at the edge node. Note that the performance with different VOQ sizes is generally the same, with the exception of the end-to-end delay performance. This is because larger VOQ size

yields longer delay between buffering and transmission. These results also convince us that with the MAN topology, a 400-packet-VOQ is enough for storing packets coming from outside of AAPN. The utilization of bandwidth can achieve 80% while keeping packet loss as low as 2%. It is worth mentioning that the performance of utilization varies queue length. We currently set it to be 80 packets, which is close to a slot size. We notice that mean end-to-end delay is relatively longer when offered load is light. This is because it takes longer time to fill out VOQ when packet arrival rate is low.

5.2 Wide Area Network (WAN)

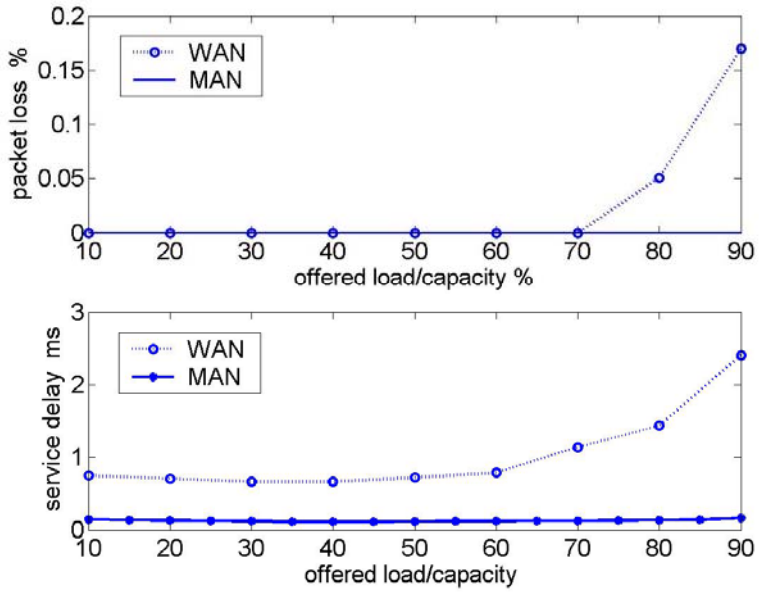


Figure 5.4: APIM performance comparison in WAN and MAN
Top panel: Packet loss rate; Bottom panel: Service delay

No packet loss occurs in the simulation of the MAN topology over the range of traffic loads shown, while packets loss occurs when offered load is around 80% in a WAN topology. Apart from the propagation delay from ingress to egress edge node, at least a round trip delay from ingress edge node to core switch is required for reservation signalling. However, the service delay shown is much smaller than the round trip time (approximately 10 ms). This improvement is due to the use of a request threshold and leftover matching featured in our scheduling algorithm. The request is made if arriving packets to VOQ exceeds the threshold. This threshold is set to be 50 packets, while a slot can in fact carry around 100 packets. Accordingly, the full request and grant delay does not

apply to the packets arriving after the threshold is reached. Moreover, the leftover matching can further decrease the grant delay because a VOQ can be served even before any request is issued for it. With VOQs served more often, fewer requests are made to the central scheduler - this decreases processing delay for each grant as shown in Figure 5.5.

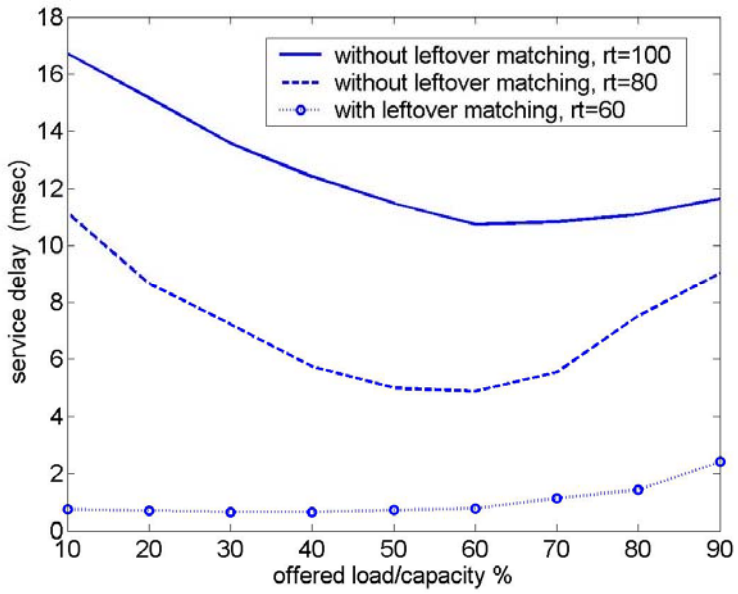
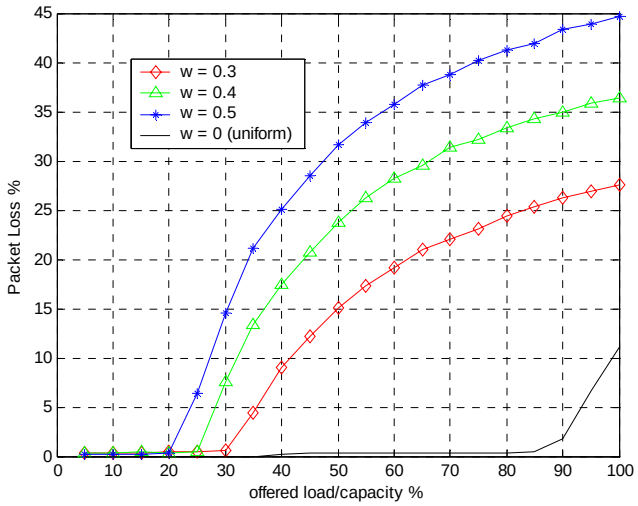


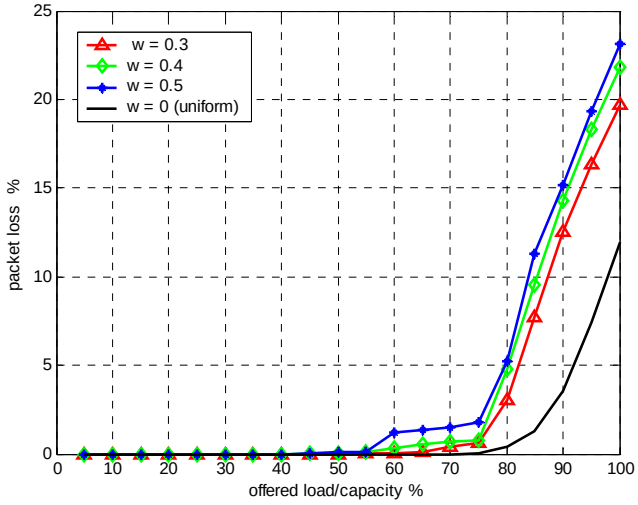
Figure 5.5 Effect of left-over matching

5.3 Non-uniform Traffic

Using the non-uniform traffic model defined previously, we run simulations for both DSA and APIM scheduling in order to compare the effect of non-uniform traffic on these two scheduling schemes.



(a)



(b)

Figure 5.6: Packet loss performance comparison between DSA and APIM: (a): Packet loss vs. offered load for DSA; (b): packet loss vs. offered load for adapted PIM

In Figure 5.6, we vary the non-uniformity parameter *w* from 0.3 to 0.5 to get unbalanced traffic. It is clear that DSA become inefficient as *w* grows. And the utilization drops by around 10% as *w* increases by 0.1. The packet loss becomes unacceptable when offered load reaches 30% of the capacity. However, the negative effect of non-uniform traffic is not so pronounced in the performance of Adapted PIM scheduling, and packet loss stays as low as 2% even when offered load reaches 75% of the capacity. Clearly, the Adapted PIM algorithm has better performance over DSA when traffic pattern is non-uniform. This is because the former calculates schedule according to the traffic demand from every edge node, while the latter allocates time slot in a static manner without knowledge of traffic demand.

5.4 Scalability Analysis

In this section, we show that, simulated under full system capacity in large networks, our design can still achieve good utilization performance with acceptable delay value. From the results of topological design for AAPN [10], we know that MAN topology consists of around 300 edge nodes, while WAN topology involves around 1000 edge nodes. In the previous sections, we studied the scheduler performance within 8-edge node network. However, as the network size scales up, more buffering is required at the edge node, this brings about larger buffer delay, which may result in unacceptable delay performance. For this reason, in order to have a better idea of the scalability of our design, we investigate the network with larger numbers of edge nodes.

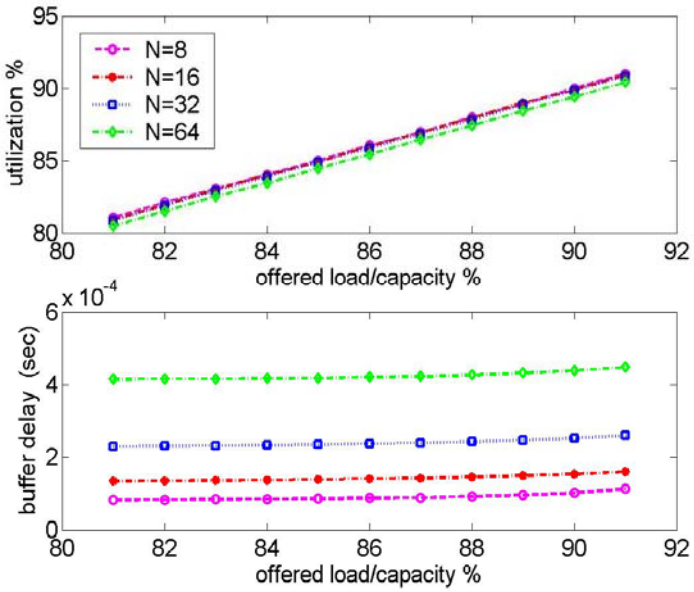
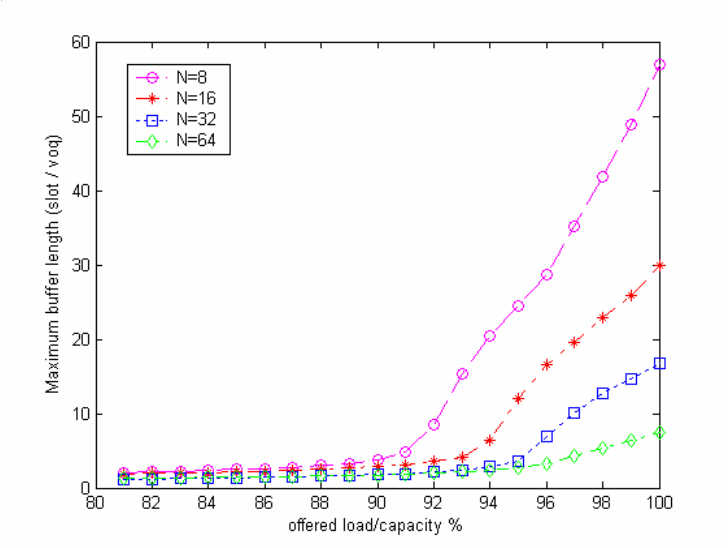
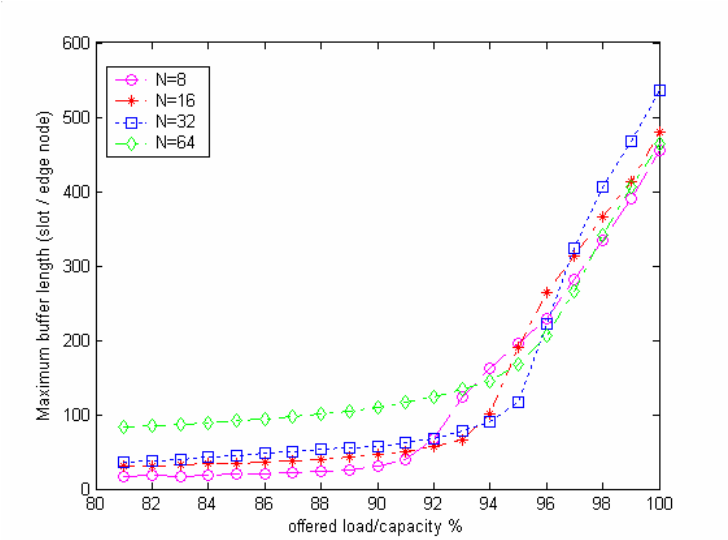


Figure 5.7: Adapted PIM performance with network size scaling up (N is the number of edge nodes in AAPN): top panel: Utilization vs. offered load; bottom panel: average buffer delay vs. offered load

Simulation results shown in Figure 5.7 tell us that, given enough buffer at the edge node, the algorithm achieves identical utilization performance with different numbers of edge nodes. Moreover, the buffer delay is growing in a more or less linear fashion as N increases. Therefore, by linear extrapolation we can expect that the buffer delay with N = 300 is about 2 msec, which is still an acceptable delay value.



(a)



(b)

Figure 5.8: Tracking of the buffer usage at the edge node

(a): Buffer usage for each VOQ; (b): Buffer usage for each edge node

Tracking the buffer usage at the edge node is applied to collect the maximum amount of buffered traffic. Notice that in Figure 5.8, the traffic is measured in slots, which means the equivalent amount of traffic that can be transferred in a whole time slot. Figure 5.8 (a) shows the size of buffer required for a single VOQ. As the number of edge node grows, incoming traffic spread out evenly among more VOQs. Given a certain amount of

incoming traffic, the demand for a single VOQ buffer decreases. In Figure 5.8 (b), for different numbers of edge nodes the demands for VOQ size stay more or less the same in the high offered load region.

5.5 Simulation Result for Differentiated Services

In this section, we focus on provision of QoS guarantees to two classes of traffic: Expedited Forwarding (EF) and Best Effort Forwarding (BE). EF, also known as premium service, can be used to build a low loss, low delay and low jitter end to end service. BE traffic has the same service as in the current Internet. In diffServ-enabled network, the scheduling strategy is a crucial technique for performing resource allocation and brings about service differentiation. Among the many available scheduling disciplines that can be implemented in the central controller of AAPN core switch, the strict priority (SP) discipline is perhaps the most commonly used. With the SP scheduler, packets with the highest priority will always be selected first. Only when the queues of higher priority are empty can packets of lower priority be served. Adapted PIM algorithm and Deterministic Slot Allocation are performed in SP discipline, in order to allocate time slots to both EF and BE traffics. EF and BE packets are stored in separated VOQ at the edge node. In the following figures we show the comparison simulation results for Adapted PIM scheduling for both diffServ traffics and uni-class traffic.

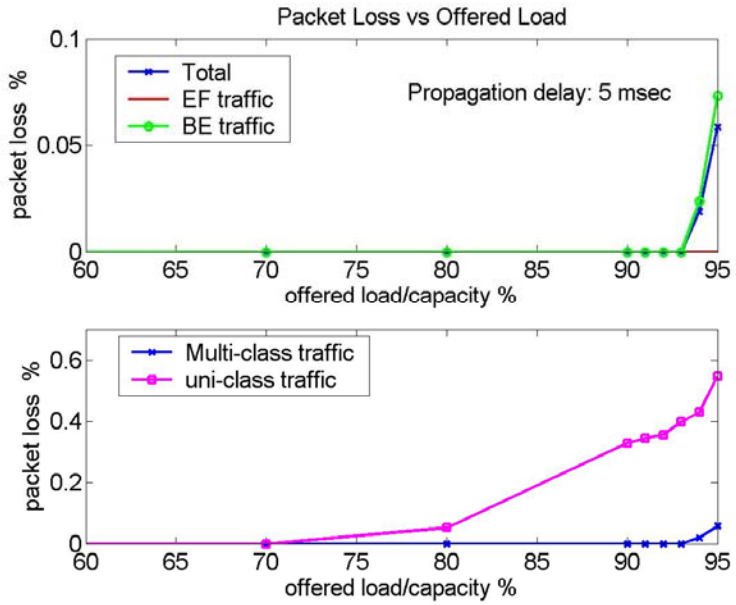


Figure 5.9: Differentiated service performance: Packet loss vs. offered load

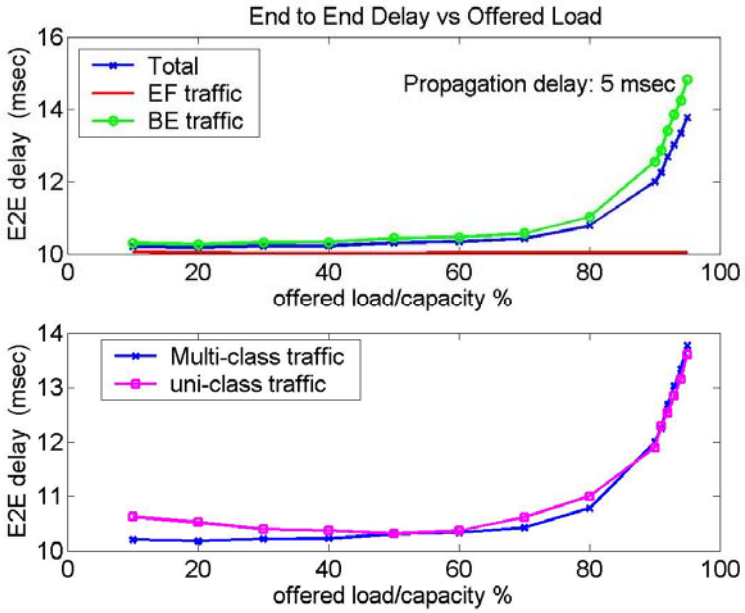


Figure 5.10: Differentiated service performance: End to end delay vs. offered load

The results shown in Figure 5.9 and 5.10 are taken from the simulation scenario with 0.005 sec propagation delay, as defined in Chapter 4, corresponds to WAN topology. We find that in Metropolitan Area Network, a single high quality BE class can provide adequate QoS at high utilization values on the order of 90%. In a WAN topology, the introduction of the EF and BE classes can increase bandwidth utilization by an order of 15% relative to a single BE class to due to the reduced loss rate at high offered load. This behaviour can be explained as follows. When a single high quality BE class is offered, the VOQ buffers must be dimensioned small enough to assure that all packets are served in time to meet the real time needs of the most stringent class. For MANs the buffer delay required for scheduling is small, as it is proportional to the propagation delay which is also small. Thus an aggregate load of 95% link capacity can be carried without overflowing the small buffer (400 packets capacity). On the other hand for WAN applications with a single BE class, the buffers must be large enough to accommodate the round trip request and grant process. At an offered load of 95% link capacity the uni-class traffic experiences a loss in excess of 0.5%, for the single class QOS buffers dimensioned to meet delay requirements of the real time traffic flows. When separate VOQ buffers are used for the EF and BE classes, one can dimension the buffers for the BE class sufficiently large to avoid overflow as there is no delay guarantee for BE traffic. The EF class is served by a small buffer sufficient for realizing delay requirements. As the EF VOQ buffer is served with non-pre-emptive priority, no buffer overflow occurs until the combined offered load

is in excess of 90%. For the uni-class case the no loss maximum utilization is only 70% in WAN applications, implying that significant bandwidth savings can be realized by providing two QoS classes, namely EF and BE instead of over provisioning for a single high quality BE class.

5.6 Comparison between Slot-by-Slot and Frame-by-Frame Scheduling

In this section, we show a joint work with Nahid Saberi, et al, on a comparison of Slot-by-Slot and frame by frame approaches [17]. These two scheduling schemes were specified, implemented and evaluated by simulation for application in WANs and MANs. For Poisson traffic, high utilization is achieved, in the order of 90%, for a single high quality, best effort transport service class. A critical distance exists where the two schemes break even in terms of end-to-end delay performance. For distances larger than this break-even value, frame-by-frame scheduling produces marginally smaller end-to-end delay than Slot-by-Slot scheduling. Thus frame by frame is suitable for WANs. The reverse is true for smaller distances typical of MANs where the Slot-by-Slot protocol yields smaller delay values.

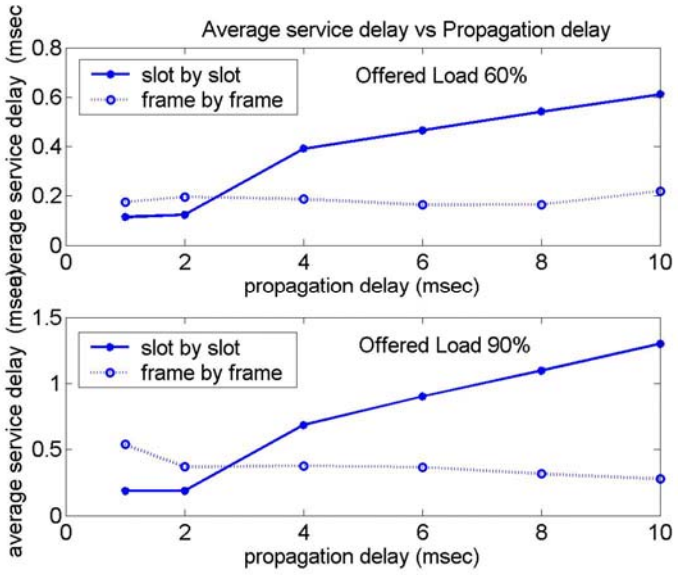


Figure 5.11: Average service delay as a function of propagation delay in uniform traffic scenario. Top panel: Offered load of 60%. Bottom panel: Offered load of 90%.

Figure 5.11 compares average service delay as a function of propagation delay for the

frame-by-frame and Slot-by-Slot scheduling methods. The delay components are propagation delay, transmission delay, and queuing delay. For simplicity, we call the latter two components service delay. It indicates that this critical network diameter is around 600km. For the uniform traffic demand scenarios no buffer overflow occurred during the simulation time. Slot-by-Slot has the bigger performance drop when topology is changed from MAN to WAN. This is because a round trip time of propagation delay is introduced by the "request" and "scheduling" phases described in Slot-by-Slot scheme.

As discussed in Chapter 2, the Slot-by-Slot scheduling has the most signaling load among all the alternative scheduling schemes in the sense that signaling is carried out on a slot basis, while frame-by-frame scheduling computes scheduling for multiple time slots within one frame all at once. Moreover, traffic arriving at the edge node is hold until receiving the grant for transmission, while in frame-by-frame scheduling a scheduler predicts the concurrent arriving traffic and schedule the transmission ahead of time to save the round trip delay spent on signaling. As a result, the latter scheduling scheme bares better delay performance in large propagation delay environment.

5.7 Conclusion

In this chapter, the scheduling approach we proposed is simulated and studied in various scenarios. We first study the effect of network size to the scheduling approaches: several algorithms are compared under Metropolitan Area Network, and Adapted PIM has shown the highest bandwidth utilization while keeping packet loss rate low. Packet loss and delay performances of Adapted PIM are compared under both MAN and WAN topology, and acceptable service delay is shown in the simulation for WAN topology. Introducing the leftover matching, to some extent, decrease the service delay incurred. However, with large propagation delay, very large buffers are still required at the edge node to hold the traffic during round trip request and grant duration, which made this scheduling approach not so attractive.

We also studied the effect of non-uniform arriving traffic pattern, to evaluate the robustness of the scheduling algorithm. As the traffic pattern varies the packet loss and utilization performances of DSA are defected due to lack of precise and up-to-date traffic demand information, while Adapted PIM has shown its robustness with variation of traffic pattern, which accommodates the property of agility of AAPN.

In order to study the scalability of the slot-by-slot scheduling approach, we simulated the network with up to 64 edge nodes. The growth of buffer delay is observed to be linear

as the number of edge nodes grows. From the simulation results, the buffer delay is around 0.4 msec when 64 edge nodes are deployed. We expect in a real network, the buffer delay is around 2 msec, which still meets the QoS requirements.

Simulation for differentiated services scheduling shows that in MAN topology, a single class of traffic can provide adequate QoS at high utilization values, even at 90% of system capacity offered load. However, in a WAN topology, the introduction of the Expedited Forwarding and Best Effort classes can increase bandwidth utilization by an order of 15% relative to a single BE class to due to the reduced loss rate at high offered load. This indicates a good alternative for scheduling in WAN topology.

Last but not least, we show the comparison of two scheduling schemes taken from the joint work with Nahid Saberi, et al. A critical break-even distance is observed from simulation with varied network propagation delays. For distances larger than this break-even value, frame-by-frame scheduling produces marginally smaller end-to-end delay than Slot-by-Slot scheduling. Thus frame by frame is suitable for WANs. The reverse is true for smaller distances typical of MANs where the Slot-by-Slot protocol yields smaller delay values.

Chapter 6

Conclusion and Future Work

6.1 Summary and Discussion

In this thesis, time-slot based scheduling for Agile All Photonic Network is studied. First, several time-slot based scheduling schemes are discussed and compared. In particular, a bipartite matching based scheduling algorithm is proposed for the Slot-by-Slot scheduling scheme, which guarantees the time slot deliveries, eliminate traffic contention at the core switch and improve the bandwidth utilization. Apart from the original features in the bipartite matching algorithm, such as low complexity and fast convergence, we further introduce the queue monitoring at the edge node and weighted leftover matching at the central scheduler to make the algorithm fit for large propagation delay environments. In addition, the simulation framework of time slot based AAPN is implemented in the OPNET simulation environment for algorithm performance evaluation purpose.

We evaluate the scheduling algorithm for the case, where a single high quality best effort service class supports all offered traffic types. In the Metropolitan Area Network, small buffers are employed at the edge nodes to provide acceptable delay performance for an appropriately small designed level of buffer overflow. The Slot-by-Slot scheduling approach is shown to be robust to variations in traffic distribution and can achieve high bandwidth efficiency with acceptably low buffer overflow probability. Accordingly it may be suitable for Metropolitan Area Network applications. The Deterministic Slot Allocation scheduling method is less robust to variations in traffic demand distribution as one would expect, however it avoids the need for signaling and reservation at the time slot level. As a result, it yields better delay performance than is possible with the Slot-by-Slot scheme. With different topology settings and arriving traffic patterns, we investigate the scheduling algorithm's capability of improving bandwidth utilization while keep packet loss rate essentially low. Simulation results show that the scheduling algorithm performs well in terms of packet loss and bandwidth utilization while keeping packet delay sufficiently low to meet real time QoS requirements. Our main contribution is to show that Slot-by-Slot

scheduling is appropriate for MAN topology with low packet loss rate while keeping the service delay meet real time QoS requirement. We also showed that the scheduling approach is robust as traffic pattern varies. Moreover, it has been shown to be scalable as the network size expands in terms of the service delay incurred by signaling between edge node and core node.

For the case where two classes of traffic are employed, a differentiated service strategy is proposed based on the Slot-by-Slot scheduling. Two classes of traffic are characterized as Expedited Forwarding (EF) traffic and Best Effort (BE) traffic respectively. We find that in a Metropolitan Area Network, a single high quality BE class can provide adequate QoS at high utilization values on the order of 90%. In a WAN topology, the introduction of the EF and BE classes can increase bandwidth utilization, and as higher as 15% of utilization can be achieved than a single class due to the reduced loss rate at high offered load.

Finally, a comparison of frame-by-frame scheduling and Slot-by-Slot scheduling is carried out in a joint research work with Nahid Saber, et al. The simulation shows that a critical distance exists where the two schemes break even in terms of end-to-end delay performance. For distances larger than this break-even value, frame-by-frame scheduling produces marginally smaller end-to-end delay than Slot-by-Slot scheduling. Slot-by-Slot has the bigger performance drop when topology is changed from MAN to WAN.

6.2 Limitation and Future Work

First thing to do is to further evaluate the time and space complexity of the proposed scheduling algorithm in a hard platform. According to the preliminary results from Anton Vinokurov's algorithm performance evaluation, the APIM algorithm is tested in a UNIX kernel with single processor. The cost of CPU computation is growing exponentially as the number of edge nodes increases. Given N multiple processors operating in parallel, the computation time can be diminished by N times. We will explore the parallel execution of the scheduling algorithm on multiple processors and obtain the optimal design of the scheduling system in terms of the algorithm computation complexity.

While we have explored various aspects of the Slot-by-Slot scheduling for AAPN, there is still much work to do in the other alternative scheduling schemes. As we mention in Chapter 2, frame-by-frame and call-by-call scheduling can bring about a tolerance of large propagation delay incurred in AAPN structure. However, the effectiveness of frame-by-frame scheduling is largely depending on the precision of traffic prediction at the

edge node side. The challenge of performing call-by-call scheduling will be in the operation of establishing and tearing down the connections between OD pairs to handle the connectionless traffic injected into AAPN.

Apart from what is said above, yet, we still need to refine our Slot-by-Slot scheduling algorithm, and further evaluate the robustness of this algorithm under other traffic patterns. Our experience allows us to extend an existing network model to support other types of traffic (self-similar, for instance) as well as other non-uniform traffic distributions. By this, we mean that with the simulation framework we already have, we can change the arriving traffic pattern at the traffic source model easily, and it is possible to investigate the algorithm performance under the other traffic patterns.

Ongoing research work will also exploits the Slot-by-Slot scheduling with Deterministic Slot Allocation. Notice that, the robustness of the DSA can be improved by allowing the slot allocation to vary from frame to frame according to traffic demand. Call by Call or control driven slot allocation similar to conventional TDM switching is one obvious alternative. A data driven approach is also possible where the network updates the frame based slot allocation based on traffic measurements and forecasts.

References

1. Anderson T. E., Owicki S. S., Saxe J. B., and Thacker C. P., "High-Speed Switch Scheduling for Local-Area Networks," ACM Transactions on Computer Systems, vol. 11, no. 4, pp. 319-352, Nov. 1993.

2. A. Vinokurov, X. Liu, L. G. Mason, "Resource Sharing for QoS in Agile All Photonic Networks", accepted by OPNETWORK 2005, Washington D.C., Aug. 2005.

3. Baldine I., Rouskas G. N., Perros H. G., and Stevenson D., "JumpStart: A just-in-time signaling architecture for WDM burst-switched networks," IEEE Communications Magazine, vol. 40, no. 2, pp. 82-89, Feb. 2002.

4. Bianco, G. Galante, E. Leonardi, F. Neri, and A. Nucci, "Scheduling algorithms for multicast traffic in TDM/WDM networks with arbitrary tuning latencies," Computer Networks, vol. 41, no. 6, pp. 727-742, Apr. 2003.

5. Bochmann G.V; Hall T.; Yang O.; Coates M.J.; Mason L.G.; Vickers R. The Agile All Photonic Network: An Architectural Outline. Queen's Biennial Conference on Communications, Feb. 2004.

6. Francois J. Blouin, Andrew W. Lee, Andrew J. Lee, and Maged Beshai, "Comparison of two optical-core networks, vol. 1, No. 1, Journal of Optical Networking.

7. Kar K., Stiliadis D., Lakshman T. V., and Tassiulas L., "Scheduling algorithms for optical packet fabrics," IEEE Journal on Sel. Areas in Comm., vol. 21, no. 7, pp. 1143-1155, 2003.

8. Liew S. Y. and Chao H. J., On slotted WDM switching in bufferless all-optical networks, HOT Interconnects, Standford University, 2003.

9. Maach A. and Bochmann G.V, "Segmented Burst Switching: Enhancement of Optical Burst Switching to decrease loss rate and support quality of service", Proc. Sixth IFIP Working Conference of Optical Network Design and Modelling, Torino, Italy, Feb. 2002.

10. Mason L.G., Vinokurov A., Zhao N., Plant D. Topological Design and Dimensioning of Agile All Photonic Networks. Submitted to "Computer Networks", 2005.

11. Matthew O. Jackson, and Asher Wolinsky, "A Strategic Model of Social and Economic Networks", 44-74, Journal of Economic Theory, 1996.

12. McKeown N., "The iSLIP scheduling algorithm for input-queued switches," IEEE-ACM Transactions on Networking, vol. 7, no. 2, pp. 188-201, Apr.1999.

13. Minkenberg C., "Performance of i-SLIP scheduling with large round-trip latency," High Performance Switching and Routing Workshop, 2003.

14. Ramamirtham J. and Turner J., "Time sliced optical burst switching," 22th Annual Joint Conference of the IEEE Computer and Communication Societies, 3 (30), pp. 2030-2038, 2003.

15. Shreedhar M. and Varghese G., "Efficient fair queueing using deficit round-robin," IEEE-ACM Transactions on Networking, vol. 4, no. 3, pp. 375-385, Jun. 1996.

16. Smiljanic A., "Flexible bandwidth allocation in high-capacity packet switches," IEEE-ACM Transactions on Networking, vol. 10, no. 2, pp. 287-293, Apr. 2002.

17. X. Liu, N. Saberi, M. J. Coates and L. G. Mason, "A comparison between time-slot scheduling approaches for All-Photonic Networks", in Proc. IEEE ICICS Conference 2005, Bangkok, Thailand.

18. X. Liu, A. Vinokurov, and L. G. Mason, "Performance comparison of OTDM and OBS scheduling for agile all-photonic network," in Proc. IFIP Metropolitan Area Network Conference, Ho Chi Minh City, Vietnam, Apr. 2005.

19. Xu L. S., Perros H. G., and Rouskas G., "Techniques for optical packet switching and optical burst switching," IEEE Communications Magazine, vol. 39, no. 1, pp. 136-142, Jan. 2001.

20. Zaim A. H., Baldine I., Cassada M., Rouskas G. N., Perros H. G., and Stevenson D., "Jumpstart just-in-time signaling protocol: a formal description using extended finite state machines," Optical Engineering, vol. 42, no. 2, pp. 568-585, Feb. 2003.

21. Zang H., Jue J.P., and Mukherjee J., "Photonic Slot Routing in All-Optical WDM Mesh Networks," Proc.IEEE Globecom '99, Rio de Janeiro, Brazil, Dec. 1999.

22. Awdeh R. Y, Mouftah H. T., "Survey of ATM switch architectures", Computer Networks & ISDN Systems, vol. 27, pp. 1567-1613, 1995.

23. Chaney, T; Fingerhut, J.A, and Turner J. S., "Design of a gigabit ATM switch", Proceedings of IEEE INFOCOM 97, Kobe, Japan, vol.1, pp. 2-11, Apr. 1997.

24. Zhang, H., "Service disciplines for guaranteed performance service in packet-switching networks," Proceedings of the IEEE, vol.83, no.10, pp. 1374-1396, Oct. 1995.

25. Kupta, P.; "Scheduling in input queued switches: a survey," technical report, http://www.students.standford.edu/~panka/

26. Karol M., Hluchyj M., and Morgan S., "Input versus output queueing on a space division switch," IEEE Trans. Communications, 1987, pp. 1347-1356.

27. Kleinrock L., "Queueing systems," Wiley, New York, 1976.

28. Francis C., Claude K., and Zoubir M., "Scheduling in Real-Time Systems," Wiley, New York, 2002.

29. OPNET, www.opnet.com.

30. Nakajima T, "Resource reservation for adaptive QOS mapping in real time match," Lecture Notes in Computer Science, 1388:1047, 1998.

31. Fabio M. Chiussi and Andrea Francini, "Providing QOS Guarantees in Packet Switches," Proc. IEEE INFOCOM 98, Mar. 1998.