Hardware Implementation of the Shortest Drop Policy for Slotted Optical Burst Switching Networks

Farid Farahmand, Amir Shaikh[†]
Advanced Networks Research Lab, The University of Texas at Dallas, Richardson, Texas, U.S.A
[†]Celoxica Inc. 900 East Hamilton Avenue Campbell, CA, U.S.A
ffarid@utdallas.edu; amir.shaikh@celoxica.com

ABSTRACT

Optical burst switching (OBS) has been proposed as a competitive hybrid switching technology to support the next generation optical Internet. This paper presents a practical scheduler design, which is suitable for the core switch in slotted OBS networks. Using hardware simulations, the performance of our algorithm in terms of processing speed, scalability, and cost will be evaluated.

1. INTRODUCTION

The amount of raw bandwidth available on fiber optic links has increased dramatically with advances in dense wavelength division multiplexing (DWDM) technology; however, existing optical network architectures are unable to fully utilize this bandwidth to support highly dynamic and bursty traffic. As the amount of bursty Internet traffic continues to grow, it will become increasingly critical to develop new architectures to provide the flexible and dynamic bandwidth allocation to support this traffic. Optical burst switching (OBS) has been proposed as an efficient way to satisfy the future bandwidth requirements of such networks.

In OBS, incoming IP packets are assembled into super-sized packets called data bursts. The edge nodes transmit data bursts following a burst header packet (BHP) after some offset time [1]. Each BHP contains routing, scheduling, and packet priority level information and is processed electronically prior to its corresponding data burst arrival. Consequently, when the data burst arrives at the intermediate core switch nodes, the burst can "cut-through" the switch on the pre-assigned path with minimum processing. Thus, in OBS the data plane retains the high capacity of all-optical switching, and the control plane takes advantage of the flexibility of electronics.

Broadly speaking, similar to optical packet-switched networks, OBS networks can be divided into slotted and unslotted categories [2]. In slotted OBS networks, control and data channels are divided into fixed-size time slots. Each control slot is further divided into several BHP slots with fixed duration. The data burst can be as long as a single or a multiple number of data slot. Therefore, in a slotted transmission mechanism, the offset time as well as the duration of the data burst and its BHP, will be expressed in terms of slots.

In an unslotted OBS network, data bursts and their BHPs can be transmitted at any time and do not have to be delayed until the next slot time boundary. However, in such networks the start and end time of the BHP's associated data burst must still be specified using some arbitrary units of time. These units of time have a much finer granularity compared to time slots and are typically on the order of a clock cycle.

The node architecture and burst behavior in slotted and unslotted OBS networks are significantly different. For example, in slotted OBS networks the core node must align all incoming optical data bursts to the slot boundaries prior to allowing them to enter the switch fabric [3]. On the other hand, in unslotted OBS networks, the data bursts are passed through the switch fabric one at a time, and there is no need for data burst alignment to slot boundaries. In terms of performance, slotted and unslotted OBS networks behave differently. For example, better bandwidth efficiency can be obtained in unslotted OBS networks, however, their burst loss probability will be higher due to unpredictable burst arrival characteristics.

A major concern in both slotted and unslotted OBS networks is contention on outgoing data channels resulting in burst loss. The scheduler block, as part of the electronic control plane in the core switch, is the primary scheduling component, which manages data burst contentions and their reservation. Developing effective scheduling algorithms and contention resolution strategies are critical in order to efficiently use the bandwidth and reduce data burst loss. Various scheduling algorithms have been proposed to effectively use the available bandwidth. For example, in the Latest Available Unscheduled Channel with Void Filling (LAUC-VF) algorithm [4] a burst chooses the unused channel that becomes available at the latest time and gaps between scheduled data bursts can be utilized. Contention resolution strategies can protect high priority bursts while reducing the overall burst loss rate. A survey of different contention resolution algorithms in OBS, including the Shortest Drop Policy (SDP), has been provided in [5]. In this paper we focus on implementing the SDP, in which the lowest priority data burst with the shortest duration will be dropped in the case of a contention, in slotted OBS networks.

Furthermore, the scheduler block in the core switch must be practical for high-speed hardware implementation in terms of processing time, scalability, and cost. The scheduler's processing time is directly proportional to the complexity of the scheduler and directly impacts the IP packet end-to-end delay. The scheduler must be scalable in order to sustain system growth as new ingress/egress ports and channels are added. The primary cost of the scheduler's implementation is its memory requirements, which highly depends on the complexity of reservation algorithms. Many efforts have been made to propose practical design solutions to OBS signaling protocols [6] and channel scheduling algorithms [7].

The contribution of this paper is to present a practical design of a scheduler block which is suitable for an OBS core node. We focus on three objectives: first, the design must be generic such that it can operate with any burst reservation algorithm; second, the design must be such that it can fully be implemented in hardware and involves no software; finally, the design must be realizable on an available off-the-shelf reconfigureable hardware device, i.e. a field programmable gate array (FPGA) operating at hundreds of MHz.

In developing the scheduler block we assume a Shortest Drop Policy for contention resolution and modeled its design using a hardware description language. We then test the functionality of the scheduler using hardware simulation. In the subsequent sections we first describe the basic concepts and system architecture of the OBS network. Then, we focus on the architecture of the scheduler block embedded in the core switch node. Finally, we describe the actual hardware prototype design of our proposed scheduler. This includes the formal description of the scheduler's internal blocks and hardware implementation of the packet scheduling algorithms.

2. OBS SYSTEM ARCHITECTURE

The OBS network consists of a set of edge and core nodes. The traffic from multiple client networks is accumulated at the ingress edge nodes and transmitted through high capacity DWDM links over the core. Edge nodes provide legacy interfaces such as Gigabit Ethernet, ATM, and IP, and are responsible for data burst assembly and disassembly. Data burst assembly refers to the process by which incoming packets are aggregated into data bursts. The reverse operation is referred to as the disassembly process.

A general description of edge nodes in OBS networks has been provided in [4]. Figure 1 shows the generic core switch node architecture in OBS. In this hybrid architecture the core switch is fully transparent to optical data bursts, while the control channels are converted into electrical signals. Each ingress link is initially demultiplexed and all data and control channels are separated. Individual optical channels are monitored for optical characteristics, including the optical power level and signal-to-noise ratio.

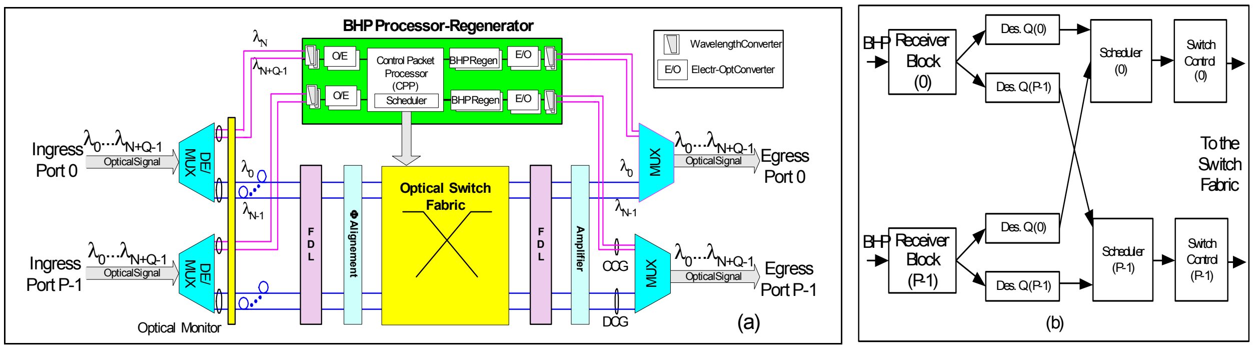


Figure 1. (a) Typical architecture of the OBS core switch node with optical data burst alignment capacity; (b) Control Packet Processor.

In the core node, the incoming BHPs on control channels are processed and regenerated in the *BHP processor-regenerator* block (BPRB). In the BPRB the BHPs are first converted into electrical signals and then sent to a *control packet processor* (CPP), where they are processed and scheduled if proper resources are available. If a BHP request was successfully reserved, the switch fabric setup needs to be updated as the corresponding data burst arrives and leaves the switch. Furthermore, each accepted BHP must be regenerated with the updated information and transmitted to the downstream core node. The control packet processor is considered as the main part of the core node's BPRB and contains the data burst reservation algorithm such as SDP.

3. HARDWARE IMPLEMENTATION

In this section we provide a formal description of the control packet processor prototype using the proposed distributed architecture, as shown in Figure 1(b). We also describe the hardware implementation of the SDP contention resolution algorithm, focusing on loss performance, cost, and scalability. In our prototype we assume a bufferless OBS network, where the control and data slots have the same duration. In addition, we assume each core node has P ingress/egress ports each having N data channels and a single control channel. For simplicity, we limit control packets to BHPs and cancellation packets.

As shown in Figure 1, prior to arriving to the CPP, each BHP is demultiplexed and converted from optical to digital signals. BHPs are then processed through the BHP receiver block, as shown in Figure 1(b). The receiver block first checks each BHP for proper framing and parity information to ensure the packet validity. Then, upon detecting a valid BHP frame, the required information fields (such as destination, burst length, offset, QoS, ingress channel) are extracted for further processing.

The receiver block also generates a timestamp count, CS_CNT, representing the control slot in which a BHP arrived. The timestamp generator can be a periodic counter with a maximum value greater than the sum of the maximum allowable data burst length and offset time. Therefore, each request is time stamped based on its corresponding data burst arrival time and the end of the data burst service time. The beginning of the burst arrival time (TS) is equivalent to the latest timestamp count value added to the offset time (Δ). The end of the scheduling time for data burst i, TE_i, is represented by

$$TE_i = TS_i + L(B_i) = (CS\_CNT + \Delta_i) + L(B_i),$$

where $L(B_i)$ is the length of the data burst i. Therefore, each request, BHP_i, requires a reservation from TS_i to TE_i and $L(B_i) = TE_i - TS_i$. Thus, the receiver block reformats each incoming BHP to include only the required payload and the scheduling timing information. Note that, in slotted transmission, TS and TE can be expressed in terms of data burst slots.

Depending on the BHP's destination, the receiver sends the reformatted BHP request to one of its P destination queues shown in Figure 1(b). The destination queue is a prioritized digital queue that services requests according to the start of their scheduling time and their priority. If the offset times between requests (with the same priority) vary, the request with the earliest data burst arrival time must be serviced first. On the other hand, the destination queue must also ensure service priority to requests with higher QoS. This is critical because of possible overflow that may occur in the destination queues due to having persistent traffic going to the same destination port. In addition, service starvation for low-priority requests 7 must be avoided in the destination queue.

All destination queues with the same index (0 through *P-1*) are interfaced to the same scheduler block, each of which schedules requests for one of the egress ports on the switch fabric. Thus, requests with different destinations can be scheduled concurrently. Details of the scheduler block and its interfaces are shown in Figure 2. The scheduler block contains the scheduling algorithm and is divided into four hardware blocks: *arbiter*, *processor*, *channel manager*, and *statistics accumulator*.

The scheduler's arbiter is required to control request flows into the scheduler. The arbiter ensures that requests with earlier scheduling time and higher priority levels are serviced first. When dealing with service differentiation, two critical issues must be addressed: possible service starvation for low-priority requests and fairness between requests coming from different control channels with the same destination. Various arbitration schemes can be considered to ensure service differentiation. In our design, we implement a priority Binary Tree Arbiter (BTA) scheme to control the packet flow from destination queues into each scheduler. The BTA [8] is a Mealy machine in which decisions are made based on the previous states. Studies have shown that such arbitration schemes can provide fair allocation of resources among all requests while avoiding service starvation. Other advantages of BTAs include their fast processing time and scalability.

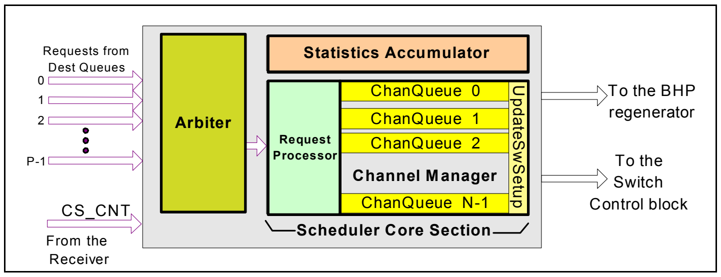


Figure 2. Details of the scheduler block and its interfaces.

The arbiter passes the requests to the processor one at a time. Based on the request's scheduling time and its duration, the processor searches through all previous reservations on different channels and checks for any available bandwidth to accommodate the new request. Obviously, an increase in the number of existing reservations for each channel results in longer search time per new request. Figure 3(a) shows the logical steps based on the Shortest Drop Policy in which the processor decides whether to grant bandwidth to a new request or not. If a request was successful, it is passed to the channel manager block. If there are no available resources, the reservation request is denied and its incoming associated data burst is discarded.

The channel manager block contains as many as *N* channel queues (one for each channel on the egress port) and an update switch setup block. The processor sends an accepted request to the proper channel queue and the request is stored until its corresponding data burst is completely serviced. The storage time for a given request *j* in a channel queue is equivalent to ($\Delta_j + L(B_j)$) data burst slots. For simplicity, and without loss of generality, in our design, we assume that all incoming requests have constant offset. In this case all requests will be sequentially stored in channel queues in order of their burst arrivals.

Furthermore, such a constraint simplifies the search, because upon the arrival of new requests, only the head of each channel queue needs to be examined. Therefore, a channel queue can be constructed using a dual-port FIFO with the appropriate depth to store the requests.

The functional description of the update switch setup block is described in Figure 3(b). When the CS_CNT changes, a new search for reservations with starting or ending timestamps similar to the current timestamp count begins through all *N channel queues*. If the starting time of a request (TS) matches the CS_CNT, then the switch control block updates the switch fabric with the new ingress-egress path. On the other hand, if the ending timestamp of an active request (TE) was found to be the same as the CS_CNT, then the switch control block must update the switch fabric and remove the active ingress/egress path. In this case, the data burst has been completely serviced through the switch fabric, and the update switch setup must remove the corresponding request from the channel queue in order to avoid its possible reactivation.

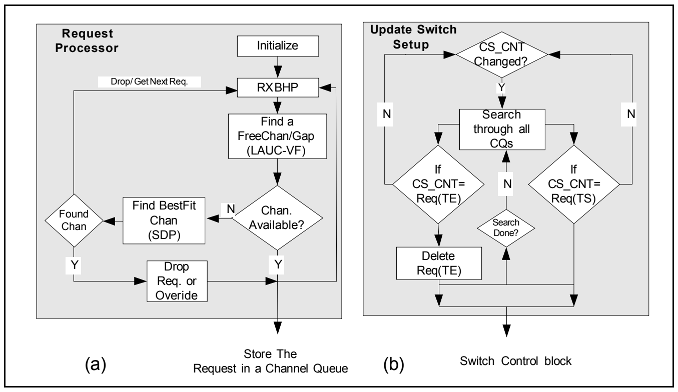


Figure 3. Functional details of (a) request processor block and (b) update switch setup block.

Soon after a reservation request is reserved, the update switch setup block sends a copy of the reserved request to the BHP Regenerator block, as shown in Figure 1, for retransmission to the next core node. The regenerated BHP must reflect the updated information, including the ingress port, and channel. The update switch setup block is also in charge of initiating other types of control packets such as cancellation packets. A cancellation packet may be required in the SDP to notify the downstream core nodes that a previous reservation has been cancelled. The statistics accumulator block can be used to keep track of the percentage of requests dropped and number of errors detected.

3.1 Scheduler Prototyping

We develop a prototype VHDL model for the control packet processor block shown in Figure 1(b). The model is implemented using a reconfigurable hardware device, i.e., field programmable gate array (FPGA), which allows easy upgrade-ability. In the design, software handling of the scheduling process is avoided and a hardware approach is taken. For practical reasons, we assume a single control channel per link and four ingress and egress ports, each having 16 channels. With a few exceptions, all blocks are modeled directly using VHDL. The scheduler core section, shown in Figure 2, including the processor (which includes the Shortest Drop Policy algorithm) as well as the destination queues, was initially modeled using the C-language. Then, the C source code was transformed into Handel-C language. Using the Celoxica DK™ design suite, the Handel-C source code was compiled and translated into a gate-level VHDL format optimized for a given targeted technology, i.e., an Altera® FPGA device. All generated VHDL blocks were combined together using a top level VHDL code. The entire design functionality was tested and verified using the Cadence® (NcSim) framework. The design synthesis was performed by Synplify® and FPGA placement and routing was done using Quartus II®.

3.2 Design Performance

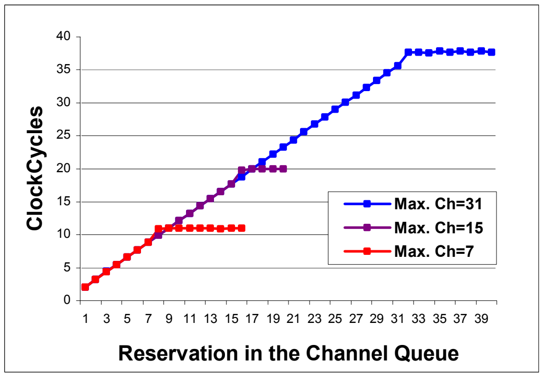
The hardware model for the control packet processor protocol was targeted for an Altera® APEX 20K FPGA family, namely EP20K400E, offering 2.5 million system gates and operating at clock rates up to 840 MHz. We assumed the operating frequency of the scheduler to be 500MHz. Figure 4(a) shows the number of cycles required to processes each incoming request. Note that, as the number of reservations stored in the channels queues becomes larger, the scheduling time for new requests increases. This is intuitive, since there will be more reservations to be verified. Once all channel queues have been saturated, the processing time reaches a steady state. Figure 4(a) also indicates that, as the number of channels on a port becomes larger, the maximum number of cycles required for a new request to be scheduled increases.

Note that the storage time for each request in the channel queue is equivalent to the sum of its offset time and duration. Therefore, the maximum memory requirement for each egress port's scheduler block will be $N\cdot(\Delta_{max}+L_{max})$, where L_{max} and Δ_{max} are the largest allowable data burst duration and offset, respectively. On the other hand, the cost of each destination queue directly corresponds to the traffic characteristics. Contrary to the case for channel queues, the memory requirement for each destination queue increases when the length of the data bursts is decreased to a single slot. This implies that each control slot will have *N* requests to be processed. Assuming persistent in which all bursts go to the same destination port for *S* continuous data slots from all ingress channels, there will be $N\cdot P\cdot S/Sr$ requests accumulated in the destination queues, where *Sr* is the average number of requests serviced per control slot. If the burstiness continues beyond *S* slots, packet overflow will occur. Therefore, the overall theoretical upper bound cost of the scheduler with distributed architecture in terms of memory requirements will be

$$[P \cdot (N \cdot (\Delta_{\max} + L_{\max})) + P^2 \cdot (\frac{N \cdot P}{Sr \cdot L_{avg}} \cdot S)] \cdot Wd,$$

where *Wd* is the width of each request (in bits) stored in the channel queue or destination queue. Note that in this case we are ignoring the memory units required for the receiver block.

Figure 4(b) shows the relative hardware cost of implementing the core section of the scheduler in the CPP using the Shortest Drop Policy. Note that, as the number of ports increases, the cost of the scheduler drastically increases.



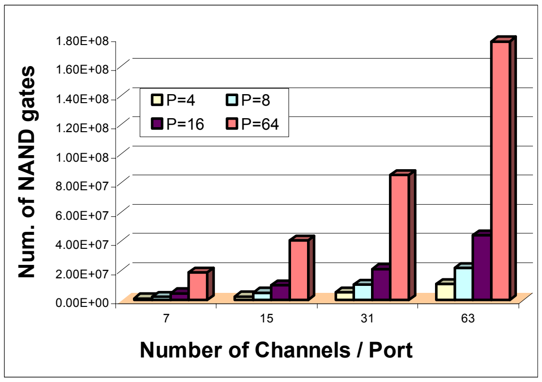


Figure 4. (a) Number of clock cycles required for each new request to be scheduled on an egress port; (b) Relative hardware cost of the scheduler core section in terms of NAND gates for various number of egress ports and embedded channels.

4. CONCLUSIONS

One important functional block in the OBS system is the control packet processor, which is a part of the control plane in the core switch node. The control packet processor is responsible for receiving BHPs and scheduling data bursts as well as controlling the switch fabric. This paper presented a detailed description of a control packet processor.

The entire header packet processor block was modeled using VHDL hardware descriptive language and implemented into a hardware programmable device such as FPGA. The performance and functionality of the design was verified using hardware simulations. The proposed design was proved to be cost efficient in terms of memory requirements, highly scalable to the system growth, and capable of operating at hundreds of MHz.

The next step in our study will be testing the control packet processor prototype under live BHP traffic and measuring statistics in terms of the number of requests dropped over a long period of time. Further hardware experiments can also provide better understanding with regards to the required storage capacity for the control packet processor block as the traffic characteristic and load vary.

REFERENCES

1. M. Yoo and C. Qiao, "Just-Enough-Time (JET): A High Speed Protocol for Bursty Traffic in Optical Networks", *IEEE/LEOS Conf. on Technologies For a Global Information Infrastructure*, pp. 26-27, August. 1997.
2. R. Ramaswami and K. N. Sivarajan, *Optical Networks: A Practical Perspective,* Morgan Kaufmann Publishers Inc., second edition, 1998.
3. S. Yao, B. Mukherjee, and S. Dixit, "Advances in Photonic Packet Switching: an Overview," *IEEE Communications Magazine*, Vol. 38 Issue: 2, February 2000.
4. Y. Xiong, M. Vanderhoute, and H.C. Cankaya, "Control Architecture in Optical Burst-Switched WDM Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 10, October 2000, pp. 1838-185.
5. F. Farahmand and J. P. Jue, "Supporting QoS with Look-ahead Window Contention Resolution in Optical Burst Switched Networks," IEEE Workshop on High Performance Switching and Routing (HPSR 2003), Italy, June 2003.
6. I. Baldine, G.N. Rouskas, H.G. Perros, and D. Stevenson, "JumpStart: A Just-in-Time Signaling Architecture for WDM Burst-Switched Networks," *IEEE Communications*, vol. 40, no. 2, February 2002, pp. 82-89.
7. S. Zheng, Y. Xiong, M. Vandenhout, and H. C. Cankaya, "Hardware Design of a Channel Scheduling Algorithm for Optical Burst Switching Routers," Proceedings of SPIE, ITCOM 2002, vol. 4872, 2002, pp.199-209.
8. K. Boahen, "A Throughput-on-Demand 2-D Address-Event Transmitter for Neuromorphic Chips," in Proceedings of the 20th Conference on Advanced Research in VLSI, Atlanta, GA, 1999.