Switch architecture for optical burst switching networks

Monther Aldwairi*[a], Mohamed Guled[a], Mark Cassada[b], Mike Pratt[b], Daniel Stevenson[b], Paul Franzon[a]
[a]North Carolina State Univ., Dept. of Electrical and Computer Engineering, Box 7911, Raleigh NC, 27695;
[b]MCNC, P.O. Box 12889, Research Triangle Park, NC 27709

ABSTRACT

We present a new switch architecture for optical burst switching networks that utilizes the just-in-time signaling protocol. Signaling is done out of band, with signaling messages undergoing electrooptical (EO) conversion at every hop while data, on the other hand, travels transparently through the configured path. The switch was implemented and deployed in advanced technology demonstration network (ATDNet); it is capable of processing 2.88 Gbps of signaling messages traffic on an Altera FPGA.

Keywords: Optical burst switching, just-in-time signaling, switch architecture

1 INTRODUCTION

The exponential growth of bandwidth demand is driving the need for routers that operate at increasing bit-rates (OC192 to OC768) and that have a very large number of ports (10s to1000s). To meet this growing demand, routers that scale with the demand on both the bit-rate and port-count are needed. The increasing demand for bandwidth has led to the development of optical fibers that could support very high bit-rates. Dense wavelength division multiplexing (dWDM) is the preferred solution for providing higher bandwidth; dWDM increases the bandwidth of a single optical fiber by creating multiple virtual fibers, each carrying multi-gigabits of traffic per second, on a single fiber.

The demand for more bandwidth is fueled by packet switched IP traffic and the traffic generated by higher layer protocols and applications, such as the World Wide Web which is bursty in nature. Optical burst switching (OBS)[4] combines the best of optical circuit switching and packet switching and can be used to carry IP over dWDM. In OBS networks a control packet is sent first, on a separate signaling channel, to set up a connection followed by a data burst without waiting for an acknowledgement for path establishment.

This paper introduces a new switch architecture designed and implemented to demonstrate the just-in-time signaling protocol for optical burst switching networks. The switch was shipped as part of the just-in-time acceleration card (JITPAC) that was deployed in a prototype network in ATDNet in October 2002 as part of the JumpStart project[3]. The JumpStart project is a joint NCSU and MCNC effort to address control plane issues in OBS networks, specifically the specification and implementation of the just-in-time protocol. Figure 1 shows a photo of the JITPAC card; to the best of our knowledge this is the first network processor implementation for OBS networks utilizing just-in-time signaling.

This paper is organized as follows. Section 2 gives an overview of the just-in-time signaling protocol. Section 3 presents the switch architecture and describes the message flows. Section 4 analyzes the switch performance and scalability. Finally, section 5 lists the conclusions.

*mmaldwai@ncsu.edu; phone 919 513 2015; fax 919 515 2285

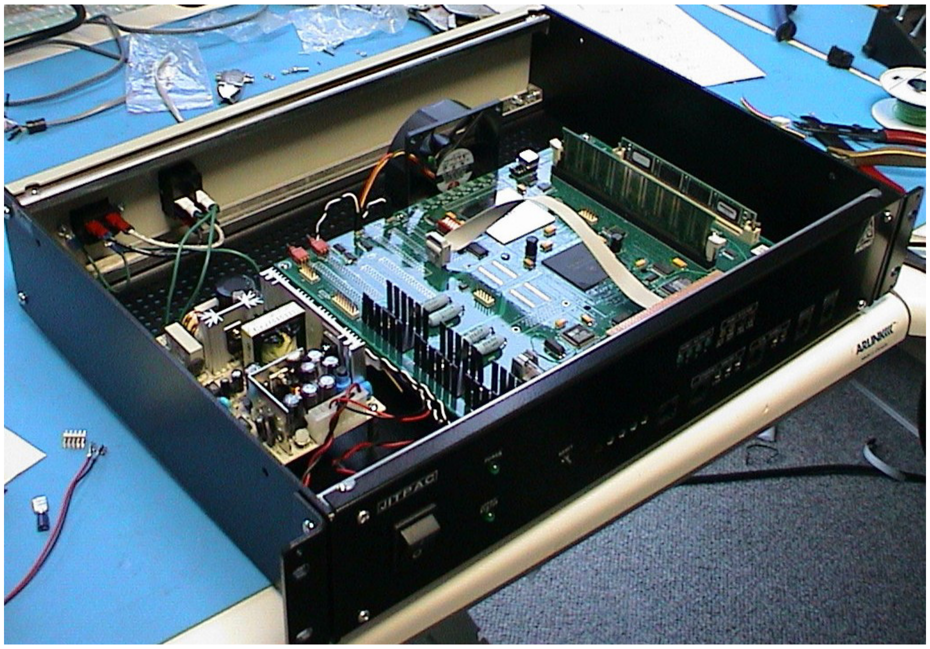

Figure 1: JITPAC card photo

2 JUST IN TIME SIGNALING PROTOCOL

Just-in-time (JIT)[5] is a reservation protocol for OBS networks that features an out of band signaling, eliminating the buffering of data bursts in intermediate switches. Signaling messages are sent ahead of the data burst to setup the intermediate switches, only the signaling messages undergo OEO conversion at every hop. The cross-connects inside the optical switches are configured before the arrival of the data burst minimizing the waiting time before the data burst is transmitted. The network infrastructure is independent of the data format and therefore, data bursts travel transparently through the configured path.

There are several ways to make reservations of data channel bandwidth[8]. Baldine et al.[6] outlined the general aspects of the JumpStart signaling architecture, the message format, the message flows, and discussed several JIT signaling schemes. Our implementation supports the explicit setup and explicit release JIT signaling scheme shown in figure 2.

Five message types are supported; SETUP, SETUP_ACK, CONNECT, RELEASE and KEEPALIVE. A SETUP message is sent ahead of the data burst by the calling host, the message is processed at every hop and the cross-connects are configured along the path to the called host. The calling switch uses a delay estimation mechanism to determine an appropriate delay value for the incoming burst and sends it back to the calling host in the SETUP-ACK message. A CONNECT message confirms the establishment of the path, however the calling host doesn't wait for the CONNECT message and it starts sending the data burst after an estimated time. The path can be torn down explicitly by a RELEASE message or implicitly through timeouts; therefore, if the burst is long the calling host maintains the path by sending KEEPALIVE messages. Any switch along the path might fail to establish the connection, due to limited wavelengths capacity or because of transmission (CRC) errors. Because FAILURE messages are not supported in hardware, the intermediate switches are left to timeout and it is left for the higher level protocol to notify the calling host.

The JITPAC supports label forwarding, where subsequent message for the same connection bypass the forwarding table lookup and are forwarded based on the labels.

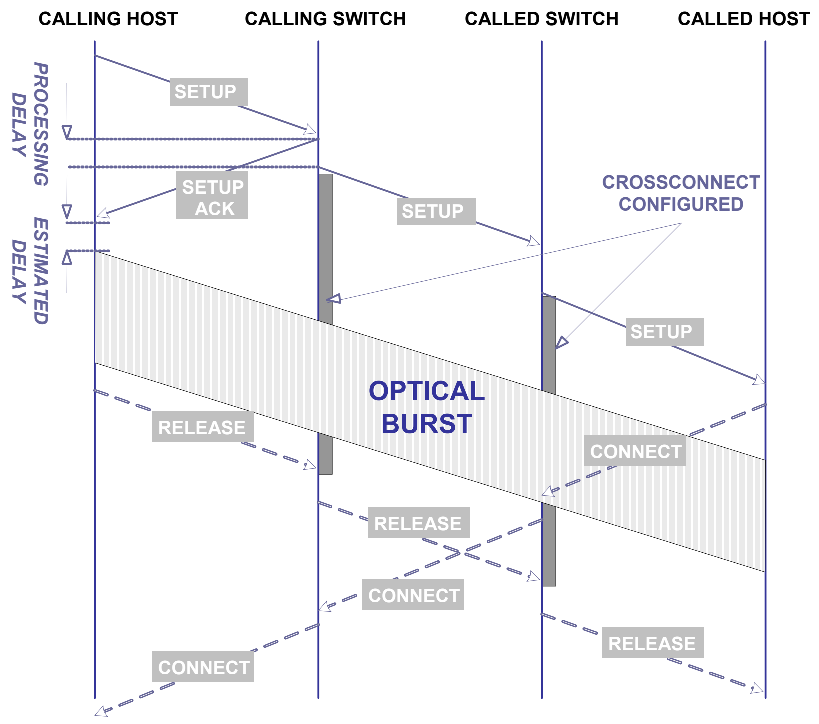

Figure 2: Explicit setup and explicit release JIT signaling

3 SWITCH ARCHITECTURE

Figure 3 shows the high level architecture of the prototype network, it has two parts the control plane and the data plane: the control plane consists of an ATM signaling channel and a JITPAC card. The data plane consists of the optical fibers and the optical burst switch element (OBSE) provided by Firstwave Secure Intelligent Optical Networks. The JITPAC card processes the signaling messages and configures the OBSE for data transmission via an Ethernet interface.

The JITPAC is a standalone development card that includes: a Motorola MPC8260 microprocessor and an Altera 20K400E FPGA connected via the 603e (64-bit) bus. The card also features a 16 MB SRAM module, an ATM copper interface and a 10/100 Ethernet port. The JITPAC card supports 4 I/O ports and 8 wavelengths per port, i.e., supporting 8 connections per port and a total of 32 connections.

The microprocessor runs the JITPAC software that is responsible for reading the incoming messages from the ATM signaling channel, configuring the OBSE through the Ethernet interface, configuring the hardware running on the FPGA and collecting statistics. The FPGA is the core of the JITPAC card; it processes the signaling messages and keeps state for the connections. It has four ingress message engines (IME) and four egress message engines (EME), connected via a switch fabric. The register address block (RAB) is a dual port RAM interfacing the JITPAC software running on the microprocessor and the message engines implemented on the FPGA.

The following subsections will explain the signaling message format, present our message engine architecture and the give examples of message flows.

3.1 Signaling message format

The signaling message consists of: a 6 bytes fixed length common header, variable length hardpath information elements (IE), variable length softpath IEs and two CRC fields. The common header consists of the protocol type, protocol version, message type, message length, softpath offset and a reserved field. There are up to 16 hardpath information elements, such as source address, destination address, call reference, label, TTL, delay estimator, wavelength and quality of service (QoS).

The microprocessor receives the message from the ATM interface, strips the softpath header and buffers the hardpath message in the RAB; Figure 4 shows the hardpath signaling message format.

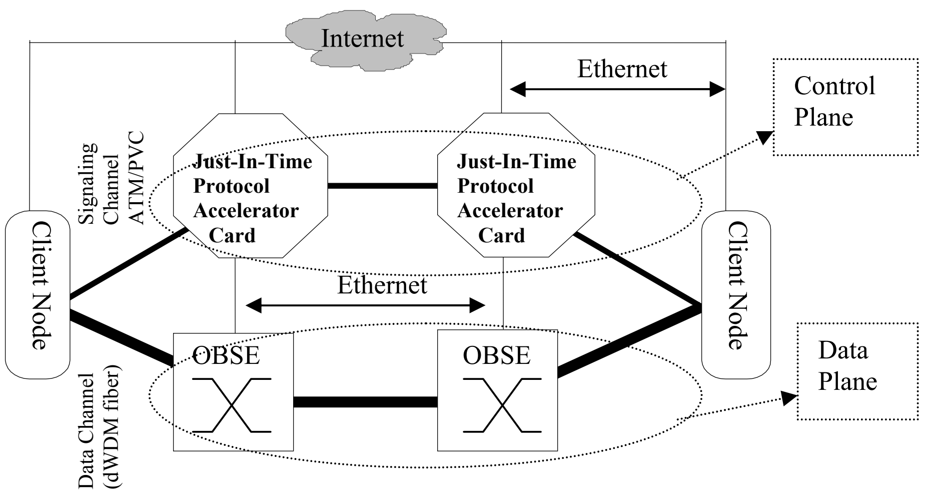
Internet
Ethernet
Control Plane
Signaling Channel ATM/PVC
Just-In-Time Protocol Accelerator Card
Just-In-Time Protocol Accelerator Card
Client Node
Client Node
Ethernet
OBSE
OBSE
Data Plane
Data Channel (dWDM fiber)

Figure 3: JITPAC deployed in a prototype OBS network

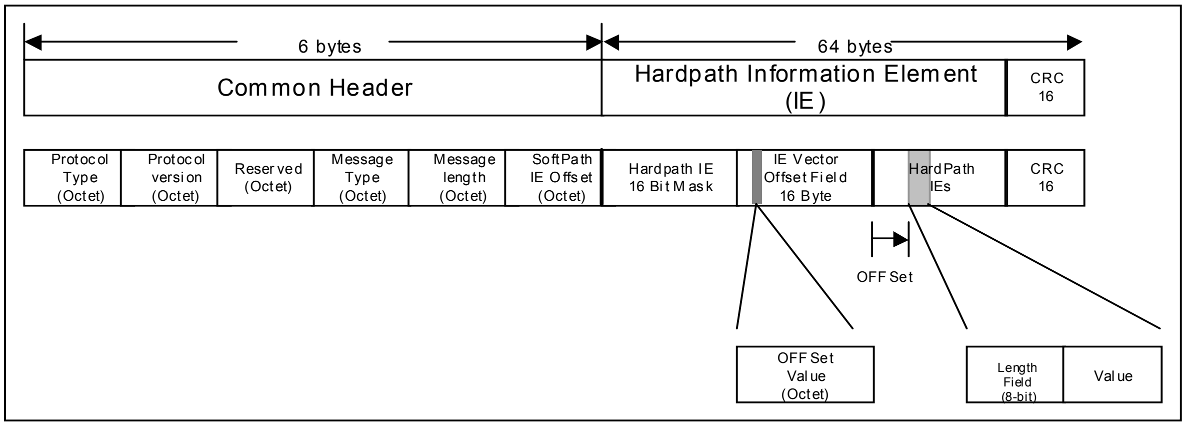

Fig 4: Hardpath signaling message format

3.2 The message engine

Figure 5 shows the detailed block diagram of one IME, the switch fabric and one EME. The IME is pipelined to increase the throughput; stage one of the pipeline consists of the IME interface that reads the message from the RAB. Stage two consists of the CRC16 checker and the message parser that parses and stores the IEs in registers. Stage three consists of four modules: the forwarding engine (FE), the state and cross-connect maintenance (SCM) module, the field update (FU) module and the acknowledgement message generator. The SCM module maintains the state machines for the connections and the cross-connect configuration. The FU module decrements the TTL and replace the input label by the output label. Stage four consists of the message reassembly module, the CRC16 generator and the output port requester (OPR) that arbitrates for the switch fabric.

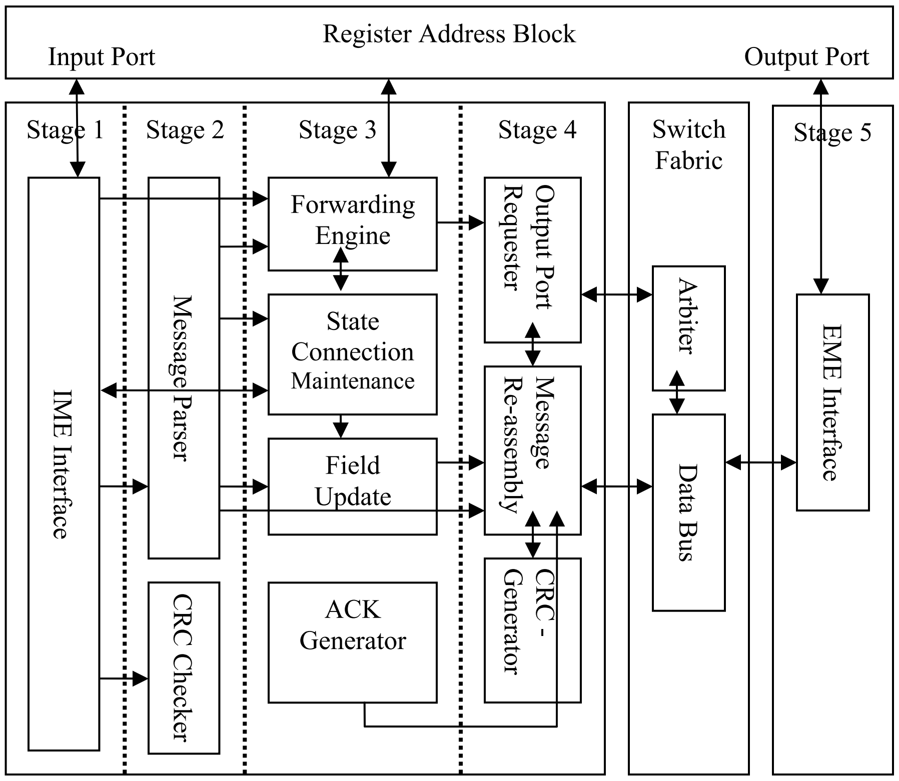

Figure 5: Message engine

3.3 Message flows

The different message types follow a similar path through the ME hardware with some differences in what IEs are parsed and how are they processed[1]. For example when a SETUP message arrives, it is first stored in the Input Message Buffer in the RAB, then the IME interface reads the message at the rate of 32 bits per cycle and feeds it to the parser and the CRC checker in parallel. The parser strips from the message the IEs that are needed by other modules and stores then in registers. The CRC checker detects transmission errors and drops invalid messages; all failures are reported to the JITPAC software where statistics of dropped messages are kept.

In stage 3, the FE reads the destination address from the parser and looks up the output port. The SCM creates a new label and associates it with the connection and the output port. The label consists of the connection number and the output port number and has link significance. The SCM reads the message type, SETUP in this case, and assigns a state machine to that connection, a maximum of 8 state machines are available per port (IME). The state machine is a simplified version of the extended FSM discussed by Zaim et al.[7], and it has three states: idle, established and error. The state machine stores the important connection attributes such as the label, output port, keep-a-live counter, etc. Through the RAB the SCM provides the processor with the information to configure the data cross connect. Finally, the FU module decrements the TTL and adds the new label to the outgoing message.

In stage 4, the updated message is reassembled, a new CRC signature is generated and appended, and the message is ready for transmission. The OPR requests a path to the output port, and once a bus is granted the message is transmitted to the EME interface and back to the RAB. The processor reads the message, appends the softpath header and sends it out on the outgoing ATM interface with the appropriate destination information.

When a KEEPALIVE or a RELEASE message arrives, the label field is used to access the FSM associated with that connection, and the output port and label are obtained. Using the label to switch the message, saves the time for table lookup and makes it possible to support quality of service in future implementations.

4 PERFORMANCE ANALYSIS

Previous work on the performance of network processors for optical burst switched networks by Mehrotra et al.[2] highlighted the importance of keeping the pipeline cycle time (T_{fwd}) as low as possible. As T_{fwd} increases, the message engine burst handling capacity decreases and the minimum burst size required increases.

Next we will identify the factors affecting T_{fwd} and the throughput of the message engine. We will also analyze the performance of the message engine in terms of area and latency.

4.1 Critical path and message engine throughput

The modules that affect T_{fwd} in our design are the forwarding engine and the scheduler. The JIT protocol utilizes a hierarchical addressing scheme with variable length addresses similar in spirit to that of the ITU telephony standard. So the problem of address lookup was reduced to simply nibble comparisons that can be executed in parallel. The forwarding table was implemented using a CAM and a RAM: The RAM stores the output ports, and the CAM stores the RAM addresses. We divide the destination address into 8 nibbles and compare them to the nibbles of the switch address. We access the CAM using the matching nibble to obtain the RAM location where the output port is stored. The forwarding engine scalability is limited by the RAM bandwidth.

The scheduler is composed of the internal switch fabric and the OPR module. Figure 6 shows the data path of the switch fabric, which consists of four 32 bits wide backplane buses, one bus per output port. The control path consists of four arbiter units, a separate unit for each bus. An arbiter unit accepts a request from each of the four IMEs and grants the bus to one of them in a round-robin approach. The grant signal is used to select which data burst is passed to the output port. This switch fabric is called multiplexer switch fabric because of the use of multiplexers. It is also called single-path network, because the same path is always followed from any given input port to a given output port.

This switch fabric allows four simultaneous direct paths between the input and output ports overcoming the internal blocking problem. External blocking (where two or more messages are destined to the same output port) is solved by input buffering, where the processor buffers the incoming messages in the RAM and passes them one at a time to the IME.

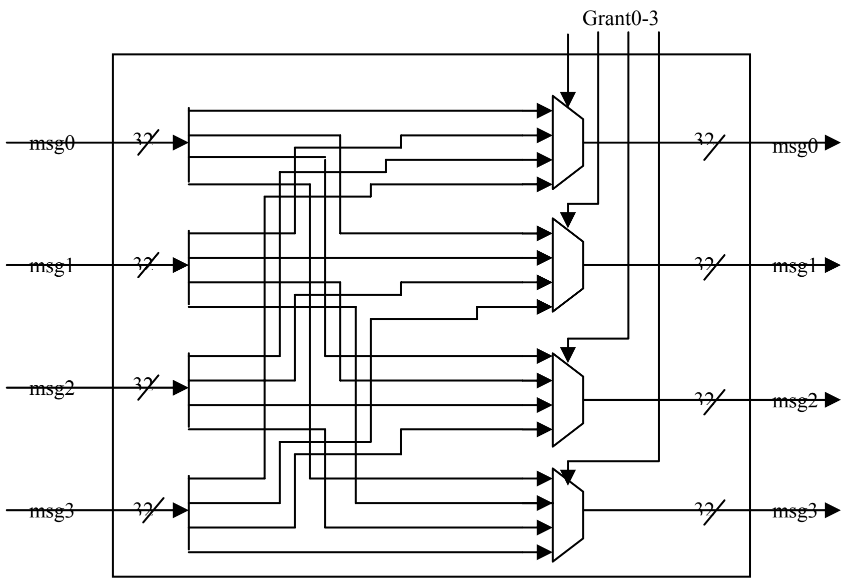

Figure 6: Internal switch fabric data path

The critical path is 20.3 ns long and passes through the switch fabric. Therefore, the switch fabric dictates the value of T_{fwd} and the maximum clock frequency (49.2 MHz). For a message size of 128 bytes this translates to a maximum throughput of 2.88 Gbps of aggregate control messages.

Module	**LE count**	**LE (%)**
Parser	729	4.4
Field update	40	0.25
State and corssconnect maintenance	1246	7.5
FE	356	2.2
8 CAMs	240	1.5
Setup_ACK	200	1.2
Total per IME	2816	16.9
Total per EME	47	0.28
Switch Fabric	556	3.3
RAB	2667	16.0
Total (4 IMEs, 4 EMEs, switch fabric and RAB)	15045	90.4

Table 1: FPGA logic elements usage

4.2 Area

The Altera 20K400E FPGA has a maximum of 16640 LEs. Our implementation of a four ports switch consumed 15045 LEs which is equivalent to 90% of the FPGA total LE capacity. Table 1 shows the detailed LE usage per module. The SCM module consumed 7.5% of the FPGA capacity; the SCM is the main control module and maintains the state for all connections. The switch fabric consumed 3.3% of the total LEs; this number grows linearly with the number of ports to be approximately 26.4% for 32 ports and 105.6% for 128 ports!

The single-path network shown in figure 6 is a simple and fast solution; it sacrifices area for speed and simplicity. It performs well for small port counts and is very suitable for the purpose of the prototype card. However, it is not scalable to large numbers of ports or high bit rates in terms of area.

4.3 Latency

Figure 7 shows that the current internal switch fabric latency scales linearly with the number of ports. The model used in the figure assumes the worst case scenario where all ports are fully loaded.

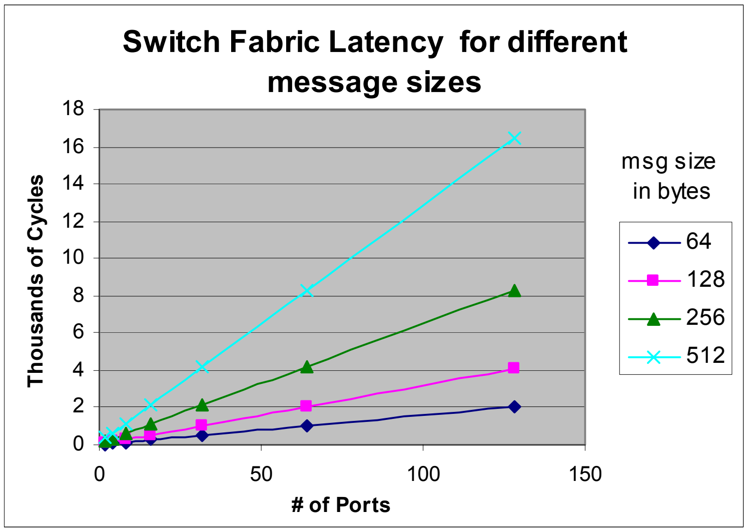

Figure 7: Switch fabric latency in cycles versus number of ports for different message sizes

For a message engine with 8 ports, and a 128 bytes signaling message, it takes 488 cycles in the worst case to pass the message though the switch fabric. Assuming a 33MHz clock; this translates to approximately 8.7 µs. For the same message size and 32 ports, the switching delay jumps to 32 µs, and to 125 µs for a 128 ports. The latency is acceptable up to 32 ports, but it is very clear that the single-path network doesn't scale well to large numbers of ports.

5 CONCLUSIONS

We implemented and deployed a new switch architecture for optical burst switching networks that utilizes the just-in-time signaling protocol. The switch achieved the throughput of 2.88 Gbps of aggregate signaling messages. The internal switch fabric is the heart of the optical switch; it is expensive in terms of area, it dictates the maximum clock frequency (throughput) and limits the switch scalability. The problem of coming up with a switch fabric that is fast and scalable in terms of area and latency is still an open one.

References

1. Mohamed Guled, "Optical network processor design for just-in-time signaling protocol message engine design", M.S. thesis, North Carolina State Univ. 2002.
2. P. Mehrotra, I. Baldine, D. Stevenson, P. Franzon, "Network processor design for use in optical burst switched networks", in *Proceedings of the International ASIC/SOC Conference*, Sept. 2001.
3. The Jumpstart Project. http://jumpstart.anr.mcnc.org.
4. C. Qiao and M. Too, "Optical burst switching (OBS) a new paradigm for an optical network", in Journal of high speed networks, Jan. 1999.
5. John Y. Wei and Ray I. McFarland, "Just in time signaling for optical burst switching networks", in Journal on Selected areas in Communications, Oct. 2000.
6. I. Baldine, G. Rouskas, H. Perros, and D. Stevenson. "Jumpstart - a Just-In-Time Signaling Architecture for WDM Burst-Switched Networks", in IEEE Communications, Feb. 2002.
7. A. Halim Zaim, Ilia Baldine, Mark Cassada, George Rouskas, Harry Perros, Dan Stevenson, "Formal Description of the JumpStart Just-In-Time Signaling Protocol Using EFSM", in *Proceedings of OptiComm02*, pp. 160-173, Boston, MA, Jul. 2002.
8. Jin-Bong Chang and Chang-Soo Park, "Efficient Channel-Scheduling Algorithm in Optical Burst Switching Architecture", in Workshop *on High Performance Switching and Routing*, 2002.