A new scheme to provide Reliable transmission in optical burst switch networks

Yong-xiang Zhao, Chang-jia Chen, Yishuai Chen
School of Electronics and Information Engineering
Beijing Jiaotong University
Beijing, China
Yongxiang_zh@sina.com, Changchen@sina.com, chenyishuai@gmail.com

Abstract—A key challenge faced by optical burst switch networks (OBS) is how to resolve burst contention that happens when two or more incoming bursts contending for the same output line (the same wavelength) at an optical switch. Most of the research effort has been devoted to reduce the burst loss probability due to contention, which cannot eliminate the loss completely. We propose an algorithm named as RCO (redundant coding optical) to provide reliable burst transmission in OBS networks. The simulation result confirms the correctness and performance of our algorithm.

Keywords—OBS; Reliable transmission; Random Coding

I. INTRODUCTION

Optical burst switch (OBS) has been a hot research topic because it has potential to utilize the huge bandwidth provided by WDM networks. A key challenge faced by OBS networks is how to resolve burst contention that happens when two or more incoming bursts contend for the same output line (in the same wavelength) at an optical switch. When a contention is happened, only one burst can be transferred, others have to be dropped.

Most of the research effort has been devoted to reduce the burst loss probability due to the contention. A brief review of this field can be found in Reference [1]. Almost all proposed techniques do not eliminate the burst loss probability completely and provide reliable burst transmission in OBS networks. In this paper, we propose a new scheme named as RCO (redundant coding optical) to provide reliable burst transmission in OBS networks.

In reference [2], we proposed an algorithm named as ROB (redundant optical burst) whose basic idea is utilizing the idle edge link bandwidth to send many duplicate copies of the original data burst to increase the probability of transfer success at core node. The RCO algorithm proposed in this paper generates the redundant data from a block of raw bursts with random coding [3], instead of the simple duplicating the original burst as redundant data in ROB. When the receiving edge can recover the block of original burst, an ack signal is sent back to the sending edge node to inform it to begin the transmission of the next block of original burst.

Reference [4] also proposed FEC to reduce burst lost probability. They use RS coding as the coding scheme. Our works are different from it in following respects: First, RCO uses the idle edge link bandwidth to transfer redundant coding burst, in contrast, reference [4] use the reservation rate to transfer the redundant coding burst, thus RCO has fewer impact to the transmission of the data burst. Secondly, RCO can guarantee the reliable burst transmission because it uses ARQ; Reference [4] can not eliminate the burst loss completely since it does not use ARQ mechanism.

In section II, we introduce the basic idea of the RCO algorithm. In section III, we describe algorithm in detail. In section IV, we evaluate it using simulation. At last, we conclude the paper.

II. BASIC IDEA

In traditional OBS, each burst only gets one chance of transmission. As an example, Figure 1(a) shows an OBS network: Edge node A sends bursts to edge node C; edge node B sends bursts to edge node D. Node E and F are core nodes. The link between the two core nodes is the bottleneck link.

Figure 1(b) plots the contention scenario: Burst 1, 2 and 3 are sent by edge node A. The time of these burst arriving at core node E is shown in first line in Figure 1(b). The second line of this figure shows arriving time of burst A and B sent by edge node B. The third line shows the burst actually transmitting on the bottleneck link between core node E and F.

Reference [4] also proposed FEC to reduce burst lost probability. They use RS coding as the coding scheme. Our works are different from it in following respects: First, RCO uses the idle edge link bandwidth to transfer redundant coding burst, in contrast, reference [4] use the reservation rate to transfer the redundant coding burst, thus RCO has fewer impact to the transmission of the data burst. Secondly, RCO can guarantee the reliable burst transmission because it uses ARQ; Reference [4] can not eliminate the burst loss completely since it does not use ARQ mechanism.

In section II, we introduce the basic idea of the RCO algorithm. In section III, we describe algorithm in detail. In section IV, we evaluate it using simulation. At last, we conclude the paper.

II. BASIC IDEA

In traditional OBS, each burst only gets one chance of transmission. As an example, Figure 1(a) shows an OBS network: Edge node A sends bursts to edge node C; edge node B sends bursts to edge node D. Node E and F are core nodes. The link between the two core nodes is the bottleneck link.

Figure 1(b) plots the contention scenario: Burst 1, 2 and 3 are sent by edge node A. The time of these burst arriving at core node E is shown in first line in Figure 1(b). The second line of this figure shows arriving time of burst A and B sent by edge node B. The third line shows the burst actually transmitting on the bottleneck link between core node E and F.
In tradition OBS networks, when burst B arrives at the bottleneck link, it has to be dropped because burst 2 from the edge node A has already reserved the bottleneck link.

In reference [2], we propose ROB to reduce the burst drop probability. The idea behind ROB is to use the idle bandwidth on edge links to provide several sending chances for each burst. As Figure 1(c) shows, we can send copies of burst B repeatedly between burst A and burst C, and we assume the core node only transfer the burst it has never transferred. Then the core node will get more chances to deliver burst B.

In this paper, we propose to use random coding to replace the simply data burst copy of ROB. When the edge link is idle, a redundant data burst is generated from a set of data bursts with random coding [3]. Let m denote the size of the data set. Any burst sent by edge node may be lost at some intermediate node due to burst contention. When the destination edge node receives m linearly independent data bursts, it can recover the data bursts set. Figure 1(d) shows an example. Suppose the data bursts set is \( \{2,3,4\} \), edge node A send sequence \( \{1,1,1,2,2 @ 1,3,2 @ 3\} \); \( 2 @ 1 \) means burst 2 xor with burst 3 in bit field. The burst 2 and 2 @ 1 is dropped by core node due to burst contention. As a result, the edge node C receive burst \( \{1,3,2 @ 3\} \), then edge node C can recover burst 2 by solve linear equation. Note in this paper, we assume core node does not record the burst it has transferred, thus the core node may transfer same burst several times.

The implementation of RCO needs resolve following key problems: Firstly, we need a buffer to store the set of data burst in the sending edge node. Then the redundant bursts can be generated from them by random coding. The main challenge is how to update the data bursts in the buffer. The simplest method is to replace the oldest burst in the buffer with the newly arriving burst. This seems reasonable. However, it does not work. For example, if current data set is \( \{1,2,3\} \), the receiving edge node receives \( \{2 @ 3, 1 @ 3\} \). In this case, the receiving edge node can solve nothing. If a newly assembled burst 4 arrives at sending edge node and replaces the burst 1 in the data set (now the data set in the sending edge node is \( \{2,3,4\} \)), the information of burst 1 will not appear in the sending burst any more. If we don’t send burst 2 or burst 3, the receiving edge node can not recover burst 1 forever and the burst 1 @ 3 is useless to the receiving edge node. In fact, we have implemented this method with simulation and find the receiving edge node got a lot of useless burst. Therefore, we have to use ARQ mechanism to explicitly inform sending edge node the time that the data burst set can be recovered at receiving edge node. Moreover, ARQ mechanism can guarantee reliable burst transmission in the OBS networks. We will discuss our method in section III in detail.

### III. ALGORITHM

#### A. Burst header

The burst header is shown in figure 2. The source/destination edge address, offset time are the standard information in the burst header of OBS network. We add two special fields for our algorithm: data set ID and ID_list. We will describe the detail of these two fields in section III.B.

```plaintext
struct burst header {
    data set ID:
    ID_list;
    offset time;
    source_edge_addr;
    destination_edge_addr;
}
```

Figure 2 2structure of burst header.

#### B. Sending edge node

ROC algorithm is composed of three parts: sending edge node, core node and receiving edge node. Figure 3 shows the structure of the sending edge node. User’s packet first arrives at burst assembler and is assembled into bursts just like in tradition OBS networks. The burst assembler includes a burst counter who records the number of bursts it generates. Every time the burst assembler outputs a burst, the burst counter increases one. The value of burst counter is recorded in the ID_list field of the burst header. This value is the unique identification of the burst. The burst generated by the burst assembler is buffered in the burst buffer 1.

Burst Buffer 2 in figure 3 buffers the data set that generates the redundant burst for transmission on the output line. Whenever the channel is idle, the burst coding block generates an output burst with random coding algorithm. It works as following: If there is a data burst in the Burst Buffer 2 that has never been sent out, this burst is selected and output to the block with algorithm of Latest Available Unused Channel void fill (LAUC-vf) [5] schedule block; if all data bursts in buffer 2 have been sent out at least once, a random number between 1 and the length of Burst Buffer 2 is generated, and denoted as x. Then x unique data bursts are randomly chosen from the Burst Buffer 2. These bursts are xor added in binary field and the obtained result is the data burst we actually send out. To convenient the receiver edge node recover the original data burst, the IDs of the bursts that generate the result data bursts is recorded in the ID_list in the header of the generated burst.

Now we describe how to update the burst buffer 2. Once the receiving edge node can recover all the bursts in the Buffer 2, it sends an ACK to the block of update algorithm in the sending edge node. After receiving the ack, the sending edge node empty its Burst Buffer 2 and loads the bursts in the buffer 1 into
the buffer2. There is a variable named as data set ID in this block to count the burst set emptied from this block. This counter uniquely identifies a burst set. The value of this counter is copied into the header of every burst sending out.

C. Core node
The core node is same as tradition OBS networks. In this paper, we use LAUC-VF to schedule burst.

D. Receiving edge node
There is a buffer in the receiving edge node to buffer the received bursts. Whenever a data burst arrives, the receiving edge node checks the data set ID field of the received burst against that of bursts in the receiving buffer. If they are not same, the arrived burst is dropped. Otherwise we continue to check whether this burst is linearly independent with the bursts in the buffer. If it is linearly independent, the arriving burst is dropped because it contains no new information; otherwise, we save it in the buffer and recover the raw data burst from the bursts in the buffer. Concretely, we solve equation (1) to check whether the arrived burst is linearly independent with the bursts in the buffer and get raw burst.

\[
P_{\text{new}} = \begin{pmatrix}
P_{\text{old}_1} \\
P_{\text{old}_2} \\
\vdots \\
P_{\text{old}_n}
\end{pmatrix} = \begin{pmatrix}
C_{\text{new}_1} & C_{\text{new}_2} & \cdots & C_{\text{new}_M} \\
C_{\text{old}_1} & C_{\text{old}_2} & \cdots & C_{\text{old}_M} \\
\vdots & \vdots & \cdots & \vdots \\
C_{\text{old}_n1} & C_{\text{old}_n2} & \cdots & C_{\text{old}_nM}
\end{pmatrix} \begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_M
\end{pmatrix} = \begin{pmatrix}
1 \\
0 \\
\vdots \\
0
\end{pmatrix}
\]  

(1)

Where, \( P_{\text{new}} \) is the burst just received, \( P_{\text{old}_i} \) is the burst in the buffer, \( x_j \) is the original data burst we need to recover. \( M \) equals to the size of Burst Buffer2 in the sending edge node. \( C_{\text{old}_ij} \) is a binary number. If the id of \( x_j \) appears in the ID list field of packet header of burst \( P_{\text{old}_i} \), then \( x_j = 1 \), otherwise is 0.

If all \( x_j \) have been recovered, receiving edge node sends ACK to sending edge node in order to inform it to begin another burst set transmission. At same time, the receiving edge node empties the bursts in the buffer.

IV. SIMULATION

A. dumbbell networks
The simulation topology is a dumbbell network as shown in Figure 1(a). Node E and F are core nodes which are connected by an optical fiber link. The link between E and F is the bottleneck link in our simulation setting. Other nodes are edge node. The links between edge nodes and the core nodes are also connected via optical link, named as edge link. We assume all optical links have the 2 wavelengths, and every wavelength has same data transmitting rate. One wavelength is used to transport control burst and the other is used to transport data burst. The maximum burst length is 8 milliseconds, and the burst assembling timeout is 20 milliseconds. The burst generated from left edge node will came to corresponding right edge node. For example, the burst generated from edge node A will flow to edge node C.

We use a wavelength bandwidth to normalize the rate, that is, the normalized rate of a wavelength links is one. We define two metrics: The bottleneck link utilization (BLU) is defined as \( BLU = \frac{R_{\text{total}}}{C} \). When the load is bigger than 0.8, the link utilization becomes stable. This is because we use acknowledge mechanism in ROC, and therefore the maximum throughput can not increase with the input load. When load is below 1.5, bottleneck utilization of ROC is apparently higher than that of OBS, meaning ROC algorithm can achieve higher link utilization than the OBS. Furthermore, with the acknowledge mechanism in receiving edge node, the receiving edge node can output burst in sequence, which guarantees the reliable data burst transmission.

The application is ON/OFF source, the time spent in ON and OFF state is exponentially distributed respectively, with an average time as 100 milliseconds. The rate during ON state is 2/10 of a wavelength rate (normalized rate is 0.2); the average rate is 0.1 for each application since the average ON and OFF time is equal. Each application is attached to a sending edge node. The rate generated from application is input into the burst assembler in the sending edge node. The destination is the corresponding receive edge node.

Varying the number of the edge links will change the net load on the bottleneck link. Note the net load is the total rate generated from all applications in the sending edge nodes.

Figure 4 is the simulation result. The x-axis is the total net load arriving at the bottleneck link. Actual rate arriving at core node is much larger than the net load because of redundant burst. Y-axis is the utilization of bottleneck link. Figure 4 shows the link utilization of ROC is improved quickly as the load of bottleneck link increases when the load is less than 0.8. And when the load is bigger than 0.8, the link utilization becomes stable. This is because we use acknowledge mechanism in ROC, and therefore the maximum throughput can not increase with the input load. When load is below 1.5, bottleneck utilization of ROC is apparently higher than that of OBS, meaning ROC algorithm can achieve higher link utilization than the OBS. Furthermore, with the acknowledge mechanism in receiving edge node, the receiving edge node can output burst in sequence, which guarantees the reliable data burst transmission.
B. Multiple hop scenario and unequal edge link bandwidth

The simulation topology is shown in figure 5. C0, C1 and C2 are core node. The other nodes are edge node. A1 and B1 are sending edge nodes. Each of A1 or B1 is attached with one ON OFF source. The burst generated from edge node A1 will flow to edge node a1. Similarly, the burst generated from edge node B1 will flow to edge node b1. So there are two bottlenecks: one is the link between C0 and C1, the other is that between C1 and C2. The other setting is same as in section IV.A. The load in two bottleneck links can be changed by varying the number of edges in the first and second stages.

![Diagram](image)

Figure 5 multihop networks

We vary the link number in the second stage from 1 to 10, and fix the link number in the first stage to 4. Figure 6 shows the simulation result. The OBS’s simulation result under same settings is also shown. X-axis is the link number in the second stage, Y-axis is the pass rate of a flow (a flow is defined as a set of burst between a pair of source and destination edge node). The figure shows the pass rate of the second stage flow of ROC is much bigger than that of OBS. This is because the flow in the second stage have more redundant burst than the flow in first stage. Specifically, since raw burst rate generated by every sending edge node is 0.1, the raw burst rate between link C0 and C1 is 0.4, so only 60% of bandwidth of the link between C0 and C1 can carry redundant burst while 90% of bandwidth of the link between B1 and C1 in second stage carry redundant burst. From the figure, the pass rate of first stage is smaller than that of OBS when link in second stage is bigger than 2. This means we should enhance the schedule strength of core node to give more priority to the flow with less redundant burst.

V. CONCLUSIONS

In this paper, we propose ROC algorithm to improve the OBS networks Qos. It uses ack mechanism to guarantee the reliability of burst transport, and uses random code to improve transport efficient. We also give out the simulation result which validates our algorithm.

REFERENCES