INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS, VOL. 8, 253–265 (1995)

SHARED BUFFER ATM SWITCH WITH DOUBLY LINKED LISTS

YEONG-FONG LIN AND C. BERNARD SHUNG
Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan 30050, R.O.C.

SUMMARY

The shared buffer memory switch (SBMS) architecture was originally proposed as an effective approach to implement ATM switch fabrics. However, in this paper we find that if an error occurs in the address chain memory of one linked list which stores the address of the next cell in the shared buffer memory, the erroneous situation will spread over all linked lists in the SBMS in a short time. In order to prevent the fault spread phenomenon, we propose two doubly linked list based architectures to combat address chain failure; these are referred to as the *Flush* and *In-Seq* schemes. The first scheme flushes the remaining cells in the faulty queue but collect their addresses for later usage. The second scheme outputs the remaining cells in their correct sequence. From our simulation, if the error injection rate is low, the performance of the *In-Seq* scheme experiences slight degradation compared with the error-free situation.

KEY WORDS: ATM; broadband ISDN; fault tolerance; shared buffer switch

1. INTRODUCTION

The telecommunication networks today are experiencing a rapid evolution. In the early eighties, the first field trials of ISDN took place, with commercial introduction in the late eighties. The volume deployment of ISDN has not yet been achieved, probably because of the lack of new attractive services and terminals. This lack of attractive services can possibly be filled by a broadband network. This network can transport many new telecommunication services, and each of these services will generate other requirements for the *Broadband ISDN* (BISDN). The large span of service requirements such as digital TV, digital HDTV, high quality videophony, high-speed data transfer, video on demand, etc. introduces the need for one universal network which is flexible enough to provide all of these services in the same way. Two other parameters are influencing the directions taken by the BISDN. These are fast evolution of VLSI and optical technology, and the evolution in system concepts such as the shift of superfluous transport functions to the edge of the network. These system concepts are made possible by the technological progress to put more functions on a chip operating at a higher speed and higher quality transmission systems. Owing to these rapid advances in technology, solutions not feasible some years ago will become economically available in the near future.

Both the need for a flexible network and the progress in technology and system concepts led to the definition of the asynchronous transfer mode (ATM) principle. This ATM concept is now accepted as the ultimate solution for the BISDN by CCITT (International Consultative Committee for Telecommunications and Telegraphy). The major characteristics of the ATM protocol include (1) no error protection or flow control on a link-by-link basis, (2) a connection-oriented operation and (3) a relatively small information field length. In ATM networks no dynamic actions are defined against packet loss. Only preventive actions are provided by ATM, by allocating resources during connection set-up and checking whether enough resources are available. Therefore, loss of cells due to queuing overflow is a typical problem. However, this momentary cell loss is controlled and limited to very small values because of the connection-oriented mode. Typical values which are considered in ATM systems range between 10^{-8} and 10^{-12} for the probability of losing a cell. In order to reduce the internal buffers in the switching nodes, and to limit the queuing delays in those buffers, the payload is kept relatively small to guarantee a small delay value and delay jitter as required by real-time services.

The shared buffer memory switch (SBMS) is one of the most attractive alternatives for the ATM switch design for BISDN telecommunications.[1,2] The switch described in this paper uses a common buffer to dynamically allocate a memory location to each output port. Favourable results have been shown under various homogeneous and heterogeneous traffic conditions[3] in terms of memory utilization efficiency. For example, the required memory space in an SBMS is only 14 per cent of the separated output buffers to achieve the same cell loss rate under bursty traffic. As illustrated in Figure 1, memory sharing in SBMS is made possible by using single-link lists for output queues. An architecture of the SBMS was proposed in Reference 4, and this is shown in Figure 2.

CCC 1074–5351/95/040253–13
© 1995 by John Wiley & Sons, Ltd.
Received August 1994
Revised May 1995

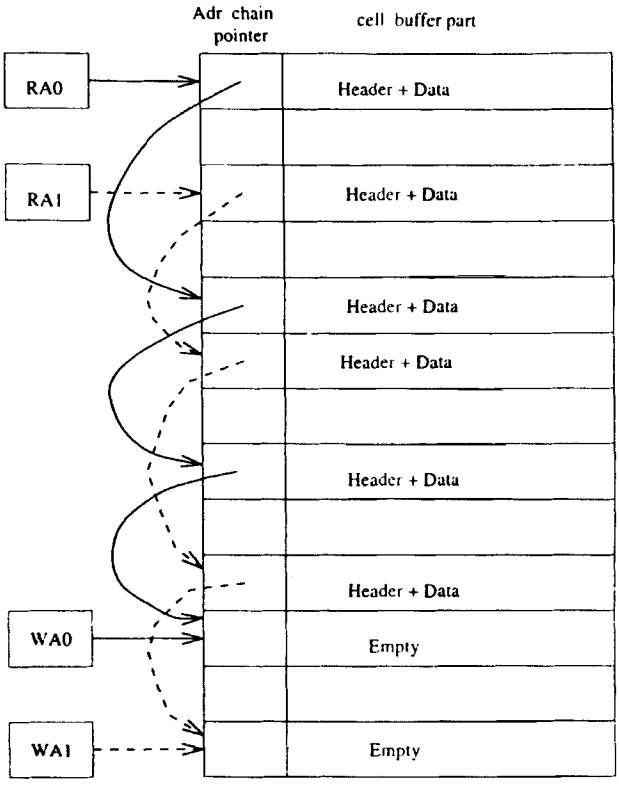

Figure 1. All cells are shared in a buffer memory by using single-link list to indicate each port. A pair of RA and WA is used to indicate the header and tail of the single-link list

In Figure 2, RA and WA indicate the beginning and the end of the address chain of one output port. When a cell is input into the switch, the write address is obtained from WA. An idle address is simultaneously read out from the idle address FIFO to overwrite WA and the next address field corresponding to the incoming cell. When a cell is read out from the switch, the reading address is obtained from RA and consequently pushed into the idle address FIFO after the reading. At the same time RA is replaced by the next address field corresponding to the read out cell.

From the viewpoint of hardware utilization, the above storage structure is quite simple and efficient. However, this storage structure is not robust because the SBMS could result in catastrophic damage if the address used to indicate the next cell is wrong. In this situation, it will read out cells which do not belong to the proper port, and the cells which are destined to the port will never be read out because of the broken link. More seriously, such an error may cause errors on other ports in the same switch. In other words, the SBMS with a single link error will propagate the errors to all other links! We refer to this phenomenon as *fault spread* and will discuss it in more detail in Section 3.

From the software point of view, a doubly linked list is more robust than a singly linked list (or linked list) because there are three fields storing the same address. For any location p in the doubly linked list, $p = p.rlink.llink = p.llink.rlink$. On the other hand, a linked list uses only one field to store the address. An algorithm has been proposed[6,7] to correct an erroneous field in a doubly linked list. This algorithm is not directly applicable in hardware because it requires multiple memory references during correction. In this paper, we propose two modified versions of the algorithm, called the *Flush* and the *In-Seq* schemes, which are suitable for hardware implementation of SBMS switches. Although the first scheme may incur an increased cell loss rate and the second an increased queuing delay, they are effective in preventing the SBMS from catastrophic situations if there is no more than one error per port during repairment. Simulation results will be presented to compare the two schemes. From these results, we contend that the *In-Seq* scheme is preferable because the performance degradation is negligible, compared with the singly linked based SBMS without link failure.

2. DOUBLY LINKED LIST ARCHITECTURES

2.1. *In-Seq and Flush schemes*

In the architecture proposed in Reference 4, two registers, RA and WA, were used to indicate the beginning and the end of the queue. If we want to implement a doubly linked list with the capacity of fault-tolerance, we have to use four registers RA1, RA2, WA1 and WA2, where RA1 and RA2 store the addresses of the first and the second cells in the shared memory, WA2 stores the address of the last cell and WA1 stores the address of the newly incoming cell, respectively. Each incoming cell is appended with two addresses. They are referred to as *rlink* and *llink*, and store the addresses of the next and the preceding cell, respectively.

The algorithm proposed in Reference 6 can be used to correct one error and detect two errors of double-link list data structures. The procedure scans the list in the forward direction until an identifier field error or forward/back pointer mismatch is detected. When this occurs, a reverse scan is initiated until an error is encountered, at which point repair is attempted. When an error is detected, the algorithm proposed in Reference 6 tries to go around the list in the reverse direction to repair the error. This is infeasible in hardware realization because it requires multiple memory references when traversing the list in the reverse direction. In the following we describe our solution to this problem.

We use the following equation to decide whether the SBMS is in normal or erroneous mode. Strictly speaking, some ports in the SBMS may be in normal and some in the erroneous mode simultaneously. Here for simplicity we assume only one erroneous port.

$$RA1 = RA2.llink \tag{1}$$

The above equation is checked when we are reading

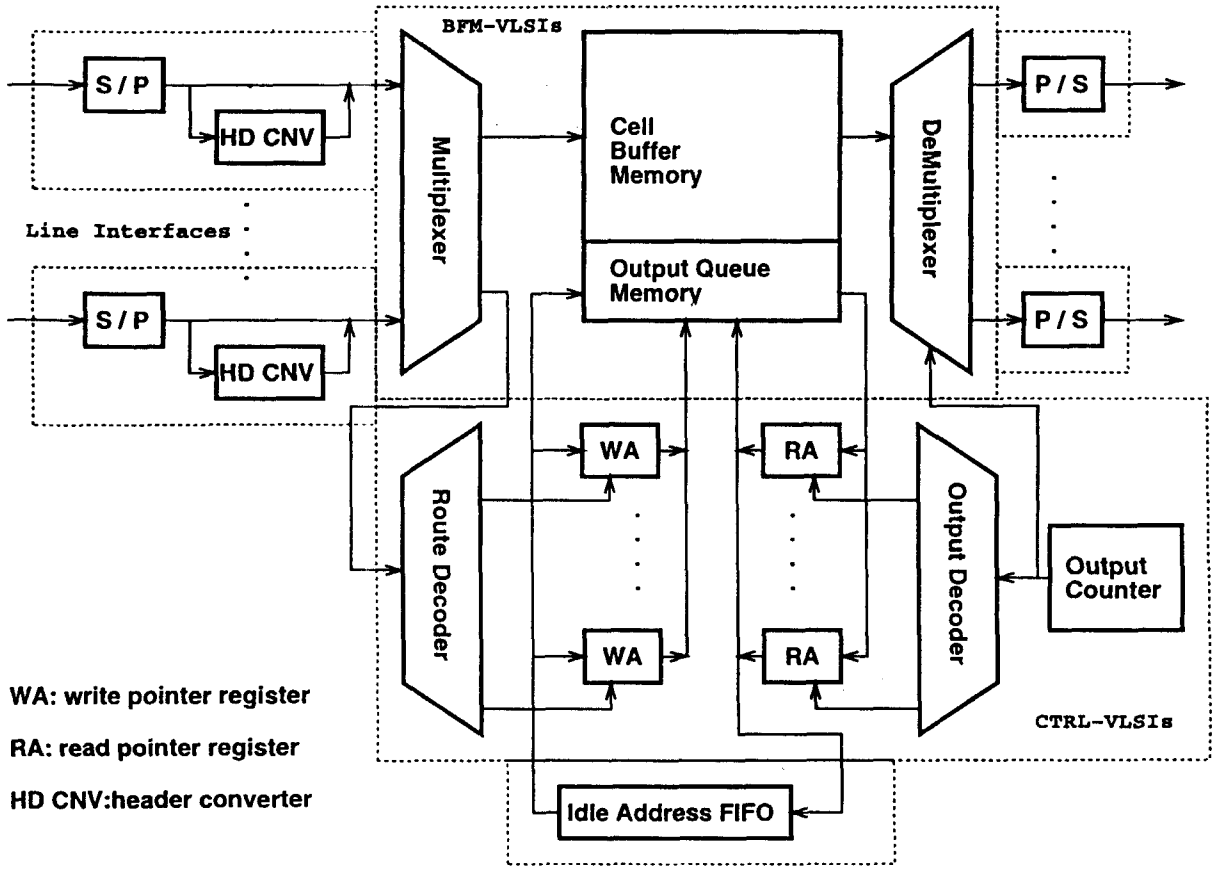

Figure 2. Architecture of the shared buffer memory switch proposed by Hitachi. Although the architecture is simple and easy for memory sharing and hardware implementation, it is not robust at all if the address for reading out is wrong

out cells from BFM via RA1. If (1) is true, the SBMS is in normal mode; otherwise, SBMS is in the erroneous mode.

In Figure 3, for example, the test of (1) is not true. After this error detection we cannot use reading address pointers RA1 or RA2 because we do not know which of the two is correct. In Figure 3 we show that the error is at the *RA1.rlink* field which causes RA2 to point to another queue. On the other hand if the error is at the *RA2.llink* field, although the contents in RA1 and RA2 are correct, the test of (1) is still not true.

Our technique is to use WA2 and WA1 (which are the only known correct addresses of the port) as the starting point to traverse the list in the reverse direction until reaching the faulty location. Owing to the memory speed limitation, we can only back traverse one location during one cell reading time.

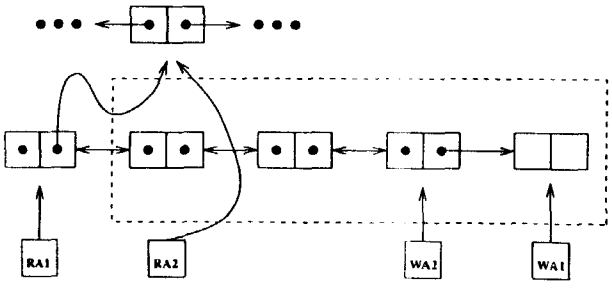

Figure 3. One error scenario in which *RA1.rlink* is wrong

During back traversing, we cannot read out cells at the same time because the read out cells will be out of sequence. It is obvious that we cannot afford to have another link error during back traversing, because in that situation we can never reach the original faulty location if the two errors are injected at the *rlink* and *llink* fields, respectively. This observation indicates that our technique can be effective only when there is no more than one error in each queue.

In Figure 3, the back traversing will stop when WA2 reaches RA1. In this case, the content of WA1 (which constantly follows WA2 during back traversing) can be used to correct RA2. On the other hand, if the error is at the *RA2.llink* field, then the back traversing will stop when WA1 reaches RA2. In this case, the content of RA1 can be used to correct the *RA2.llink*. After the error is corrected, we can now read out cells from RA2.

In this paper, we refer to the above method as the *In-Seq* scheme, because it can guarantee that all cells in the faulty queue be read out in sequence. Note that, however, there is a latency for the back traversing, and hence the performance of average queueing delay may be degraded. In fact, the cell loss rate may also be degraded in the erroneous situation because the shared buffer memory is more likely to be full.

An alternative to the *In-Seq* scheme is not to output cells after fixing the faulty queue. In Figure

3, this scheme is to give up all cells between RA1, RA2 and WA2, WA1 (in the dashed box). We therefore refer to this method as the *Flush* scheme. In this scheme, a new queue is created right after error detection, and cells entering this new queue can be output normally.

Note that back traversing is performed in the *Flush* scheme when there is no cell output. Back traversing is still important because (1) the error location needs to be fixed (2) the address of the flushed cells needs to be reclaimed to the idle address FIFO. Intuitively the *Flush* scheme should enjoy a short delay because it makes use of the free time to perform back traversing, and a worse cell loss rate due to the flushed cells. In Section 3, we will present simulation results for both schemes.

2.2. *Algorithms*

In the following *BFM* is the buffer memory, *AVL* is the idle FIFO, *r_old_cell* is the first cell to read out, *w_new_cell* is the new cell to be written in, *wp1* is the auxiliary register used in error detection for WA1, *wp2* is the auxiliary register used in error detection for WA2 and *rp1* is the auxiliary register used in error detection for RA1.

Write (In-Seq)
BFM ← *w_new_cell(address = WA1)*
llink(w_new_cell) ← *WA2*
WA2 ← *WA1*
WA1 ← *AVL*
rlink(w_new_cell) ← *AVL*

Read_Normal (In-Seq)
if *(RA1 = RA2.llink)* **then**
r_old_cell ← *BFM(address = RA1)*
AVL ← *RA1*
RA1 ← *RA2*
RA2 ← *RA1.rlink*
mode ← *normal*
else
wp1 ← *WA1*
wp2 ← *WA2*
mode ← *erroneous*

Read_Erroneous (In-Seq)
if *(wp2 = rp1)* **or** *(wp1 = rp2)* **then**
mode ← *normal*
else
wp1 ← *wp2*
wp2 ← *wp2.llink*
mode ← *erroneous*

Write (Flush)
BFM ← *w_new_cell(address = WA1)*
llink(w_new_cell) ← *WA2*
WA2 ← *WA1*
WA1 ← *AVL*
rlink(w_new_cell) ← *AVL*

Read_Normal (Flush)
if *(RA1 = RA2.llink)* **then**
r_old_cell ← *BFM(address = RA1)*
AVL ← *RA1*
RA1 ← *RA2*
RA2 ← *RA1.rlink*
mode ← *normal*
else
wp2 ← *WA2*
rp1 ← *RA1*
rp2 ← *RA2*
mode ← *erroneous*

Read_Erroneous (Flush)
if *(wp2 = rp1)* **or** *(wp2 = rp2)* **then**
if *(wp2 = rp2) AVL* ← *rp2*
mode ← *normal*
else
AVL ← *wp2*
wp2 ← *wp2.llink*
mode ← *erroneous*

2.3. *Examples*

In Figure 4 we illustrate how the *In-Seq* scheme is used. When reading out the cell from the port, we check if the condition *RA1 = RA2.llink* is true. In the above example, the error is at the *rlink* field and the checking condition is false because RA2 has pointed to the cell not belonging to the original port. If no action is taken, the port will read out the cell not belonging to the port.

At the same time, we load the values WA2 and WA1 into *wp2* and *wp1*, respectively, because those cells incoming into the faulty port after error detection will be updating the content of WA2 and WA1. In Figure 5 we use *wp2* and *wp1* for back traversing by using left link until RA1 = *wp2* or *wp1* = RA2. The condition is used when the error is at the *llink* field. Therefore it is not shown in Figure 5. When back traversing is finished, in the example we overwrite RA2 with *wp1*. Now we can use RA1 and RA2 for cell reading because the value in RA2 has been corrected. Note that while we perform back traversing we cannot afford another error in the left link. Otherwise we never reach the faulty location.

Consider now the another *Flush* scheme shown in Figure 6. When reading out the cell from the port, we check if the condition *RA1 = RA2.llink* is true. In the above example, the condition is false because RA2 has pointed to the cell not belonging to the original port. At the same time, we copy the values in RA1, RA2 and WA2 into auxiliary registers because the original registers RA1, RA2 and WA2 will be needed to record the addresses of

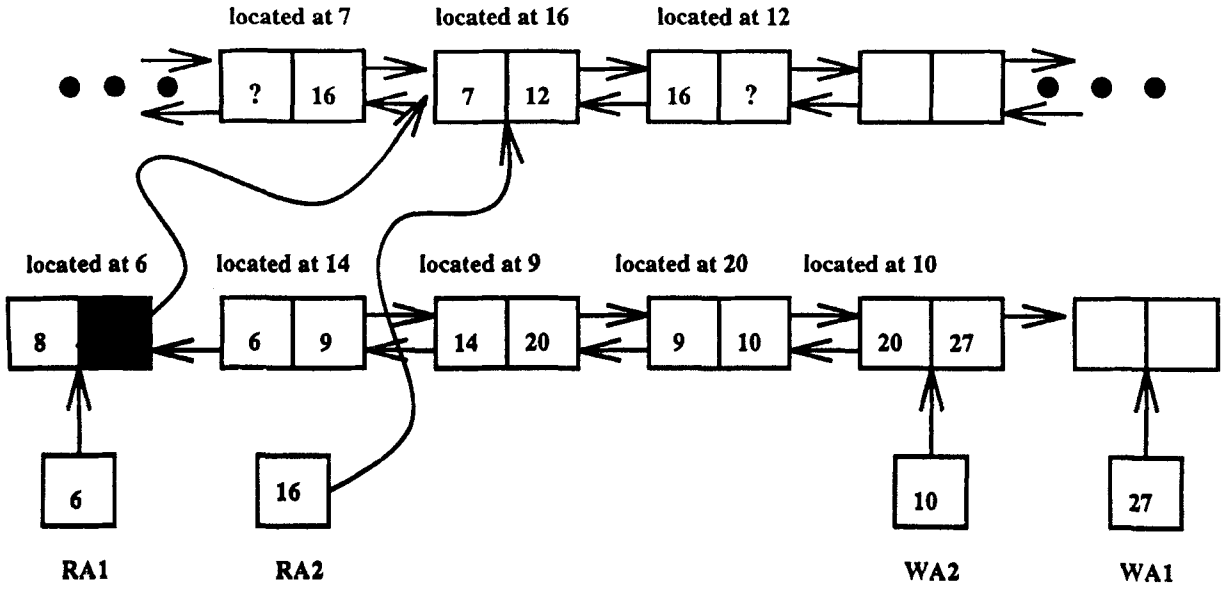

Figure 4. *Normal* mode enters *erroneous* mode in the *In-Seq* scheme

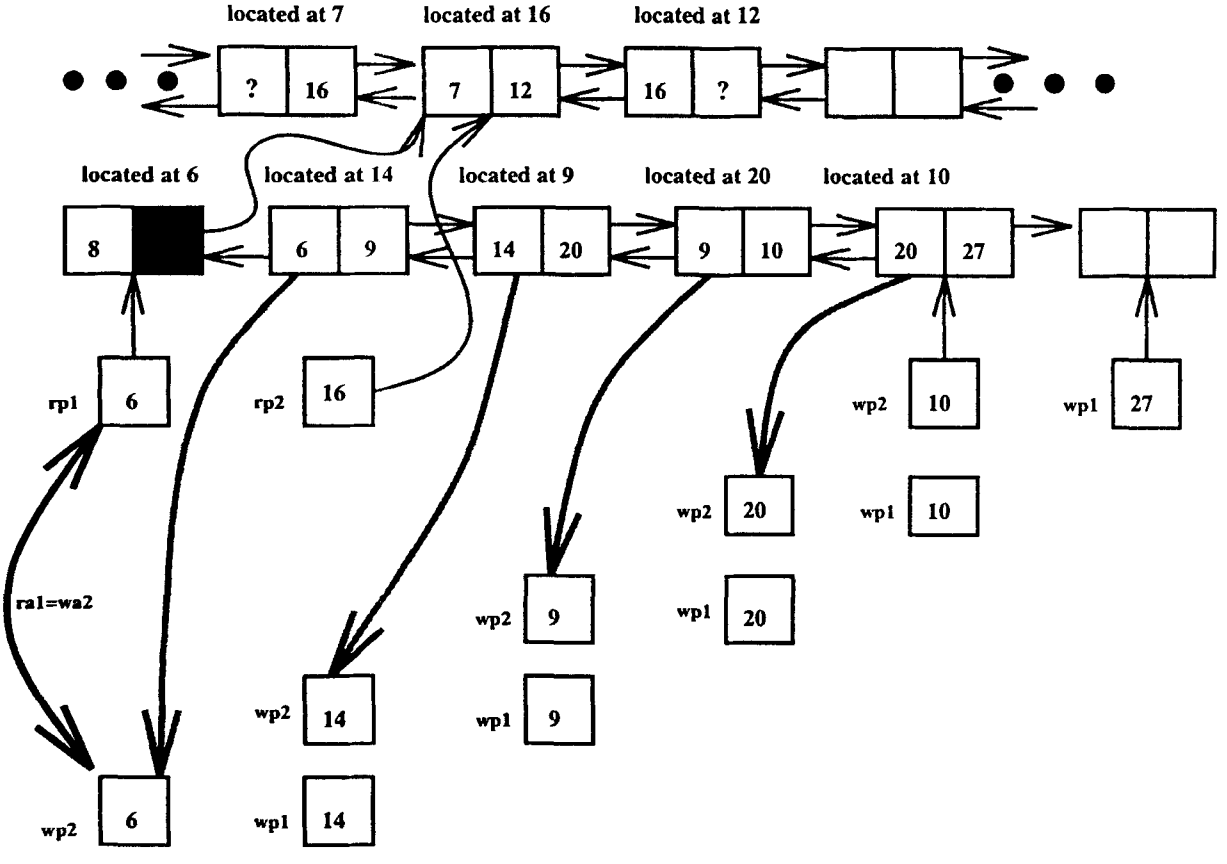

Figure 5. Back traversing in the *erroneous* mode in the *In-Seq* scheme

newly arrived cells during erroneous mode. Figure 6 shows the operation of erroneous port link repairment. We back traverse to the header of faulty port by using the register *wp2*. During traversing we collect the addresses of flushed cells for later use. If there are newly arrived cells, we output them at once. If there is no newly cell arrived, the queue performs back traversing. It stops traversing by checking if *rp1* = *wp2* is true and the port is returned to *normal* mode by using RA1 and RA2 for reading cells out.

2.4. *Architectures*

The architecture of *In-Seq* is shown in Figure 7. The detailed operations of *In-Seq* are described in

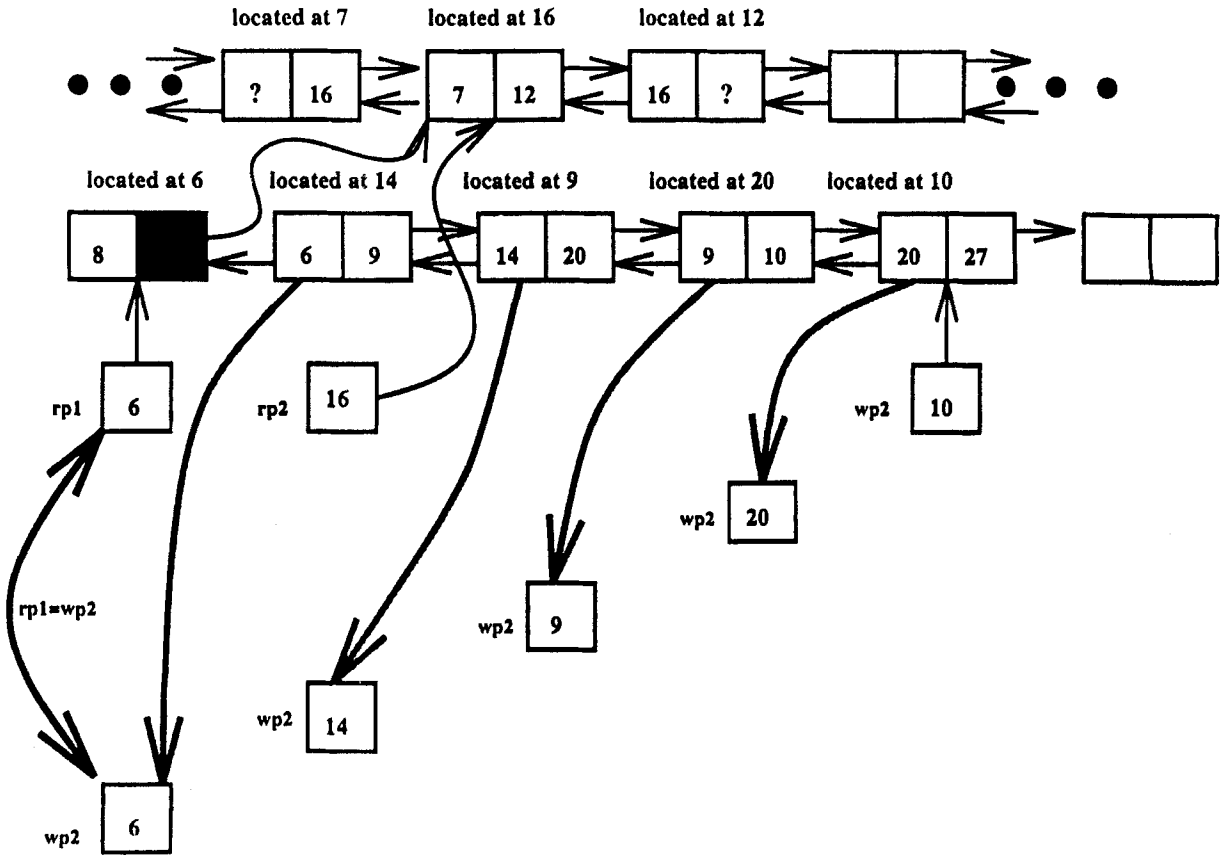

Figure 6. Back traversing in the *erroneous* mode in the *Flush* scheme

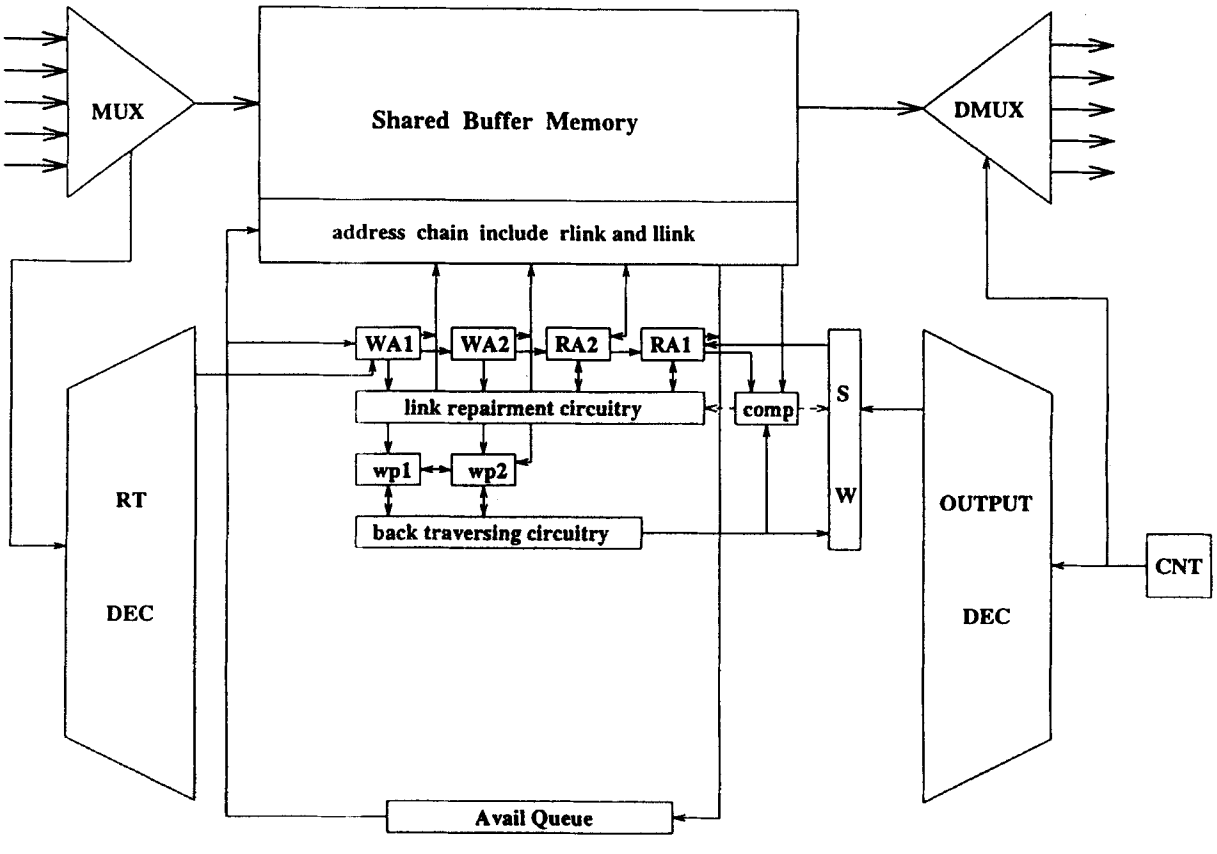

Figure 7. Architecture of the fault-tolerant shared buffer memory switch using the *In-Seq* remedy is shown. In the Figure we only show one port equipped with the capacity of fault tolerance for read and write under only one faulty link field

the following. Two auxiliary registers, *wp1* and *wp2* are required to store the contents of WA1 and WA2 at the time when an error is detected, through the *link repairment circuitry*. The functions in the link repairment circuitry include register transfer control (for example *wp1* ← WA1) and link repairment (for example RA2 ← *wp1* in Figure 5). When the port is requested to output cells during *erroneous* mode, the *back traversing circuitry* performs one back traversing per reading cell time until *wp2* = RA1 or *wp1* = RA2. When the link error is repaired, we return to *normal* reading. The switch (denoted as *SW*) is used to select the content of RA1 (*normal* mode) and *wp1* (*erroneous* mode).

In the *Flush* scheme three auxiliary registers, *wp2, rp1, rp2* are required to store the contents of WA2, RA1 and RA2 at the time when an error is detected, as shown in Figure 8. The function of *gateway* is to dump register values and form an isolation between the new created queue and the faulty queue. After error detection a new queue is created by using WA1, WA2, RA2 and RA1 to store new incoming cells. During this mode when the port is not requested to output cells, the *output decoder* through SW performs back traversing by the *recovery and read circuitry* until *wp2* = *rp2* or *wp1* = *rp1*. At the same time the addresses of flushed cells are pushed back to the idle address FIFO through recovery and read circuitry. The port can use RA1 and RA2 to read cells out before and after the link repairment.

There are two drawbacks in our techniques: (1) cells read out from the faulty port had longer delay or increased cell loss rate, (2) no more errors are allowed during reverse traversing and reading (one fault per port at any time). However, in the next section, we will show by computer simulations that the performance (delay and cell loss rate) degradation is quite small. The hardware overhead for fault tolerant design includes some multiplexers, comparators, registers and an extra address chain memory. In comparison to the catastrophic consequences of address chain faults, we consider such hardware investment to be worthwhile.

3. SIMULATION RESULTS

3.1. *Traffic model and simulation set-up*

To model switch performance quantitatively, we consider the *uniform geometrically bursty* traffic model[5] in which an input alternates between active and idle periods of geometrically distributed duration. Cells destined for the same output port arrive continuously during an active period. The duration of the active period is characterized by a parameter p. The probability that the burst lasts for a duration of i time slots (consists of i cells) is then

$$P(i) = p(1-p)^{i-1}$$

Note that we assume that there is at least one cell in a burst. The mean burst length is given by

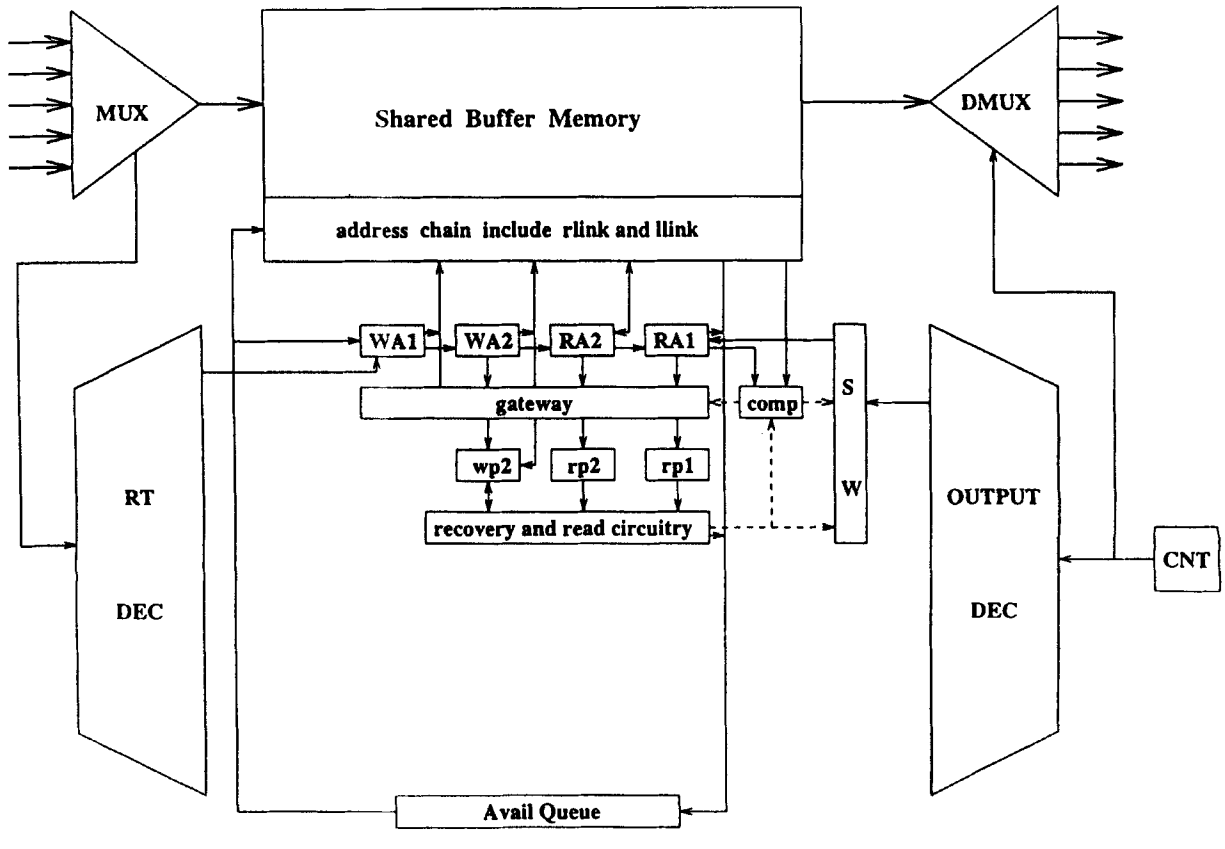

Figure 8. Architecture of the fault-tolerant shared buffer memory switch using *Flush* remedy is shown. In the Figure we only show one port equipped with the capacity of fault tolerance for read and write under only one faulty link field

$$E_B[i] = \sum_{i=1}^{\infty} iP(i) = \frac{1}{p}$$

The idle period is geometrically distributed with parameter q. The probability that an idle period lasts for j time slots is

$$Q(i) = q(1-q)^j$$

Unlike the duration of an active period, the duration of an idle period can be 0. The mean idle period is given by

$$E_I[i] = \sum_{j=0}^{\infty} jQ(j) = \frac{1-q}{q}$$

Given p and q, the offered load ρ can be found by

$$\rho = \frac{E_B[i]}{E_B[i] + E_I[j]}$$

We assume that there is no correlation between different bursts, and that the destination of each burst is uniformly distributed among the outputs. Note that the uniform random traffic model is a special case with $p = 1$ and $q = \rho$.

Our simulation experiments measure the queuing delay and cell loss rate of an 8×8 shared buffer memory switch. Queuing delay is measured as the duration between when the cell is written into the SBMS and when it is read out. Cell loss rate is measured by the ratio of the number of lost (due to flushing or buffer overflow) cells to the number of total cells. Simulated cells that arrive at the SBMS will be discarded or lost when there is no available address in the idle address FIFO. The statistics of about 10^7 cells are collected (over all queues in the switch) for each simulation. Thus, a loss probability smaller than 10^{-7} cannot be measured and a loss probability below 10^{-5} could contain only a few errors. If needed, however, loss probabilities below 10^{-5} can be extrapolated based on our results. Average burst lengths of 1 (the uniform random traffic) and 8 are considered. If the offered load and buffer size must be fixed, the load is 0·9 under uniform and bursty traffic and the buffer size is 256, respectively. Error injection probabilities are 10^{-6} and 10^{-5} and the faults are only injected into the address chain memory, not the registers in the queuing management circuits.

3.2. *Simulation results and interpretations*

Figure 9 shows the *fault spread* duration versus load. In single-link based SBMS, we choose one port randomly and inject an error into the next address of one cell in this queue. Normally each address in BFM is either in the idle address FIFO or in the address chain memory of some queue. If there were two addresses in the address chain memory with the same content due to a fault, the queue management of the SBMS will later produce two copies of the same address in the FIFO. When these two copies are used in some other two address chains, this will generate a new fault and the phenomenon will spread all over the SBMS. Finally it results in a catastrophic situation because every port will output the cells that do not belong to the proper port. The duration from the first erroneous port to the last erroneous port is simulated and shown in Figure 9. From Figure 9 the duration will decrease as the load is increased and we see that after about 100 cell slot times, all eight ports will be in erroneous situations.

When errors are injected under bursty traffic, Figure 10 shows the queuing delay versus load of these two schemes. At a higher error injection rate of 10^{-5}, the *Flush* scheme experiences less queuing delay compared with the *In-Seq* scheme as load increases. At a lower error injection rate, 10^{-6}, the difference is negligible among the *Flush, In-Seq* and no error cases.

When errors are injected under bursty traffic, Figure 11 shows the cell loss rate versus load for these two schemes. At the higher error injection rate of 10^{-5}, the *Flush* scheme experienced higher cell loss rate compared with the *In-Seq* scheme as load increased. At the lower error injection rate of 10^{-6}, the difference is negligible among the *Flush, In-Seq* and no error cases for loads ranged from 0·8 to 0·99. Notice that cell loss rate of the *In-Seq* scheme is almost the same under error injection rates 10^{-5} and 10^{-6}.

When errors are injected under uniform traffic, Figure 12 shows the queuing delay versus load of these two schemes. At the higher error injection rate of 10^{-5}, the *In-Seq* scheme experiences a slightly higher queuing delay compared with the *In-Seq* scheme with load ranging from 0·9 to 0·97. The other four curves almost collapse together even though different remedies are attempted with different error rate.

When errors are injected under uniform traffic, Figure 13 shows the cell loss rate versus load for these two schemes. Under uniform traffic with buffer size 256, there is almost no cell loss when the load is below 0·97 for the no error case and the *In-Seq* scheme. On the other hand, the *Flush* scheme has experienced some cell loss.

For short-term performance degradation which is of concern for transmission impact on a particular connection, it is desirable for our schemes to introduce little fluctuation at delay or cell loss in order to maintain quality of service (QoS). In Figures 14 and 15, we show the delay and cell loss performance of the two schemes within a small window (100 or 1000 time slots) after error injection. It can be seen that the *In-Seq* scheme is faster to leave the fluctuating situation than the *Flush* scheme.

In summary, we have investigated the performance measures when distinct remedies are used

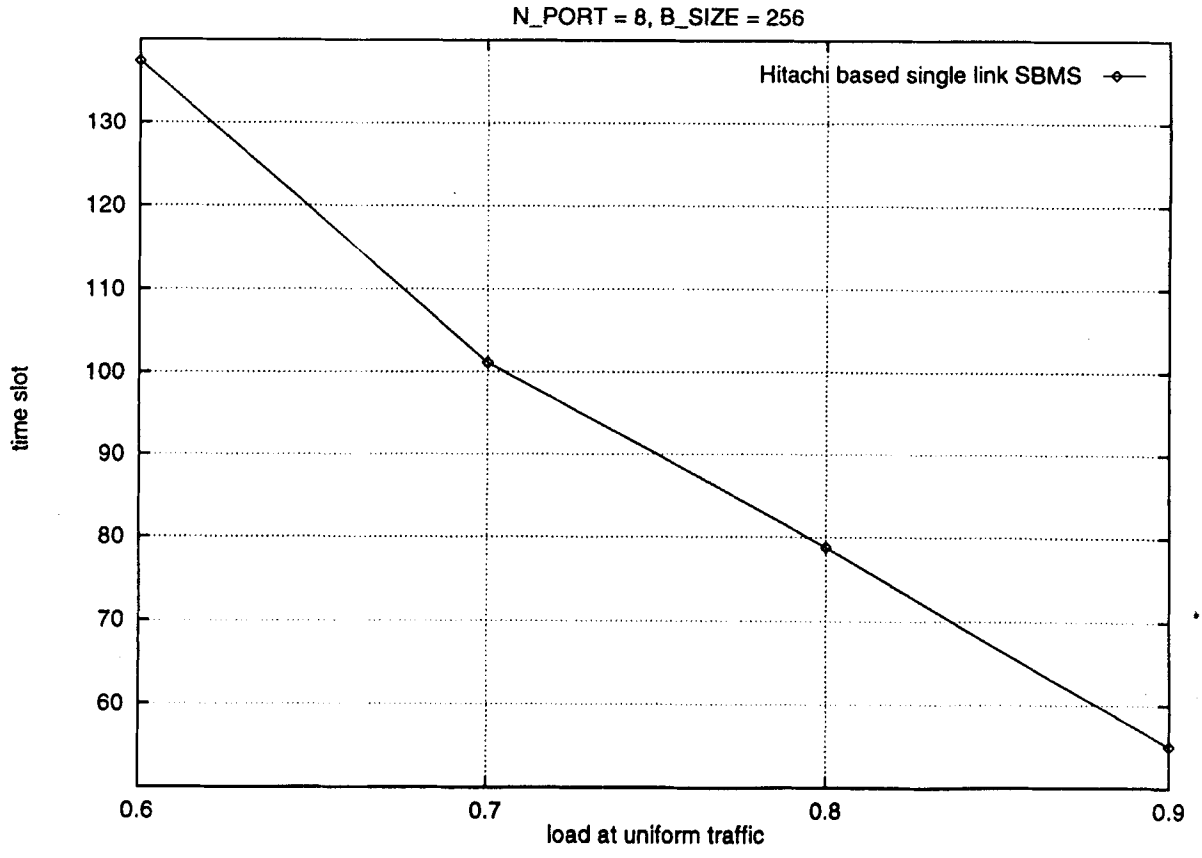

Figure 9. The fault spread duration versus load at uniform traffic

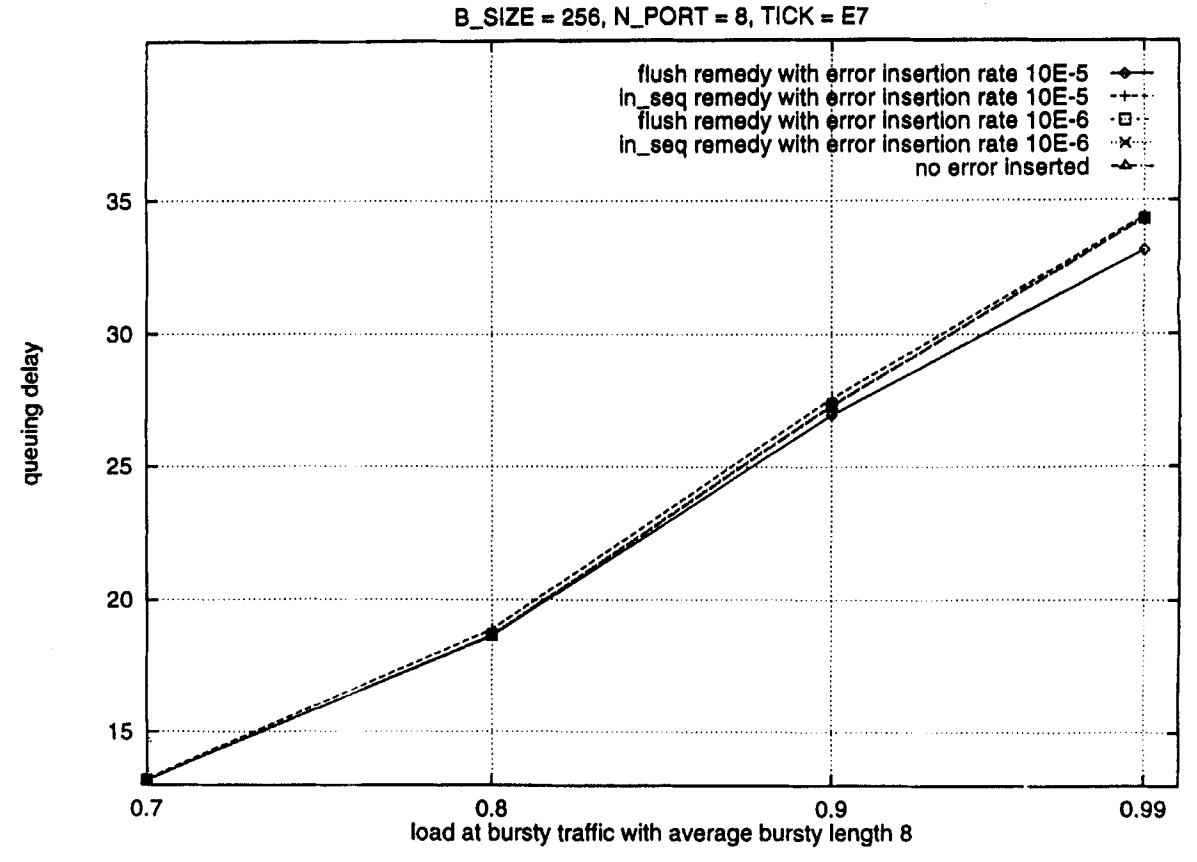

Figure 10. Queuing delay versus load at bursty traffic

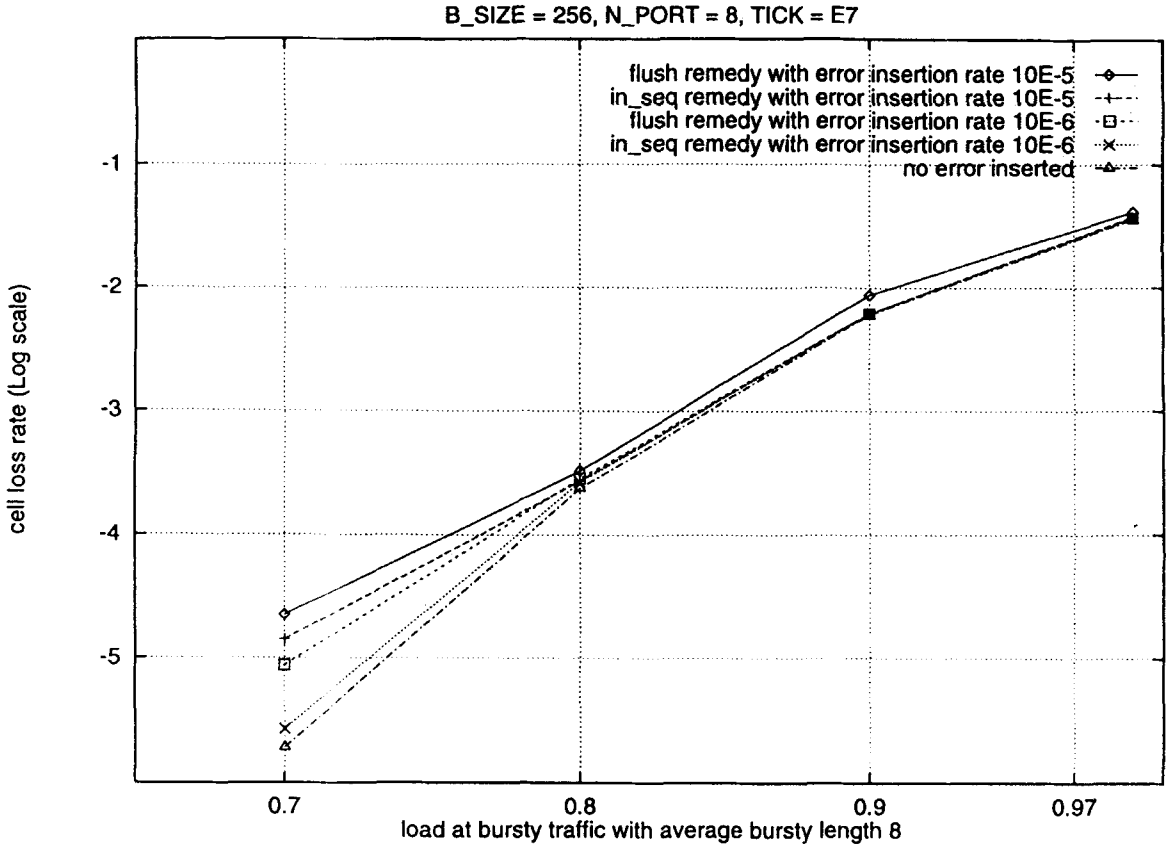

Figure 11. Cell loss rate versus load at bursty traffic

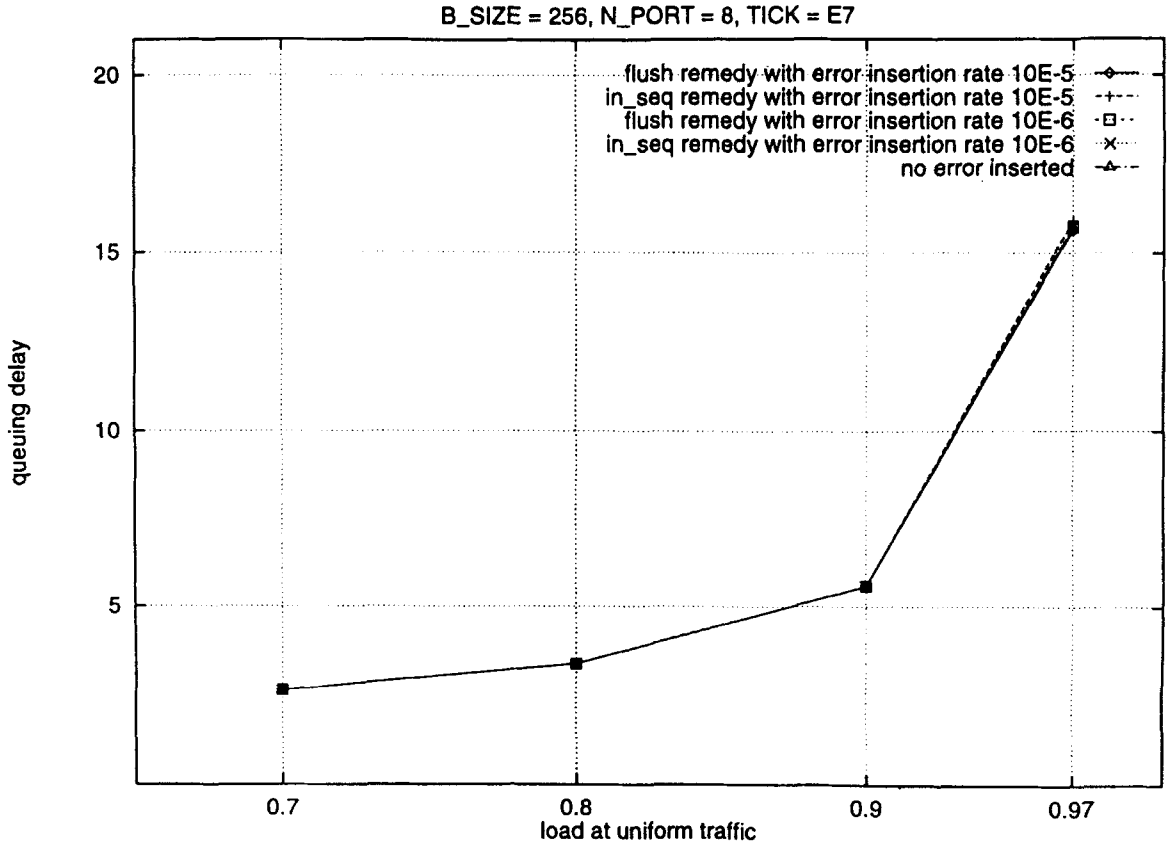

Figure 12. Queuing delay versus load at uniform traffic

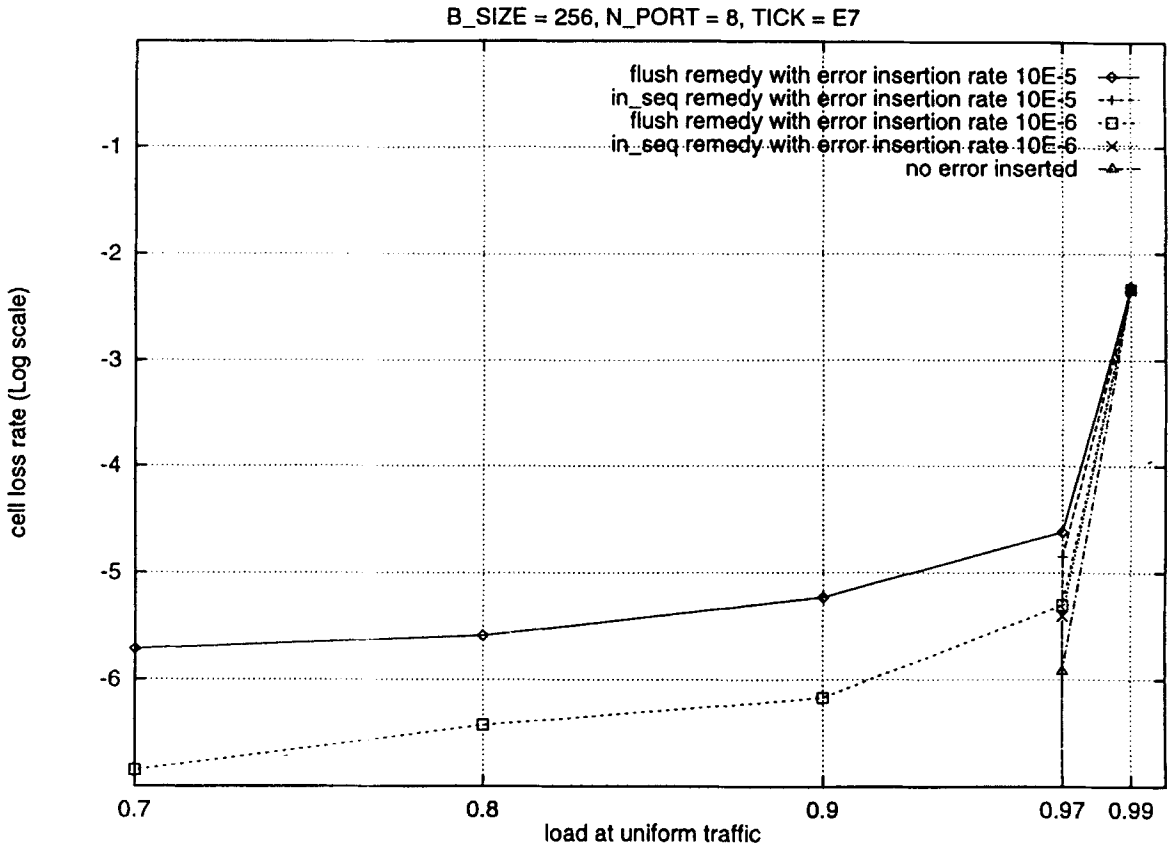

Figure 13. Cell loss rate versus load at bursty traffic

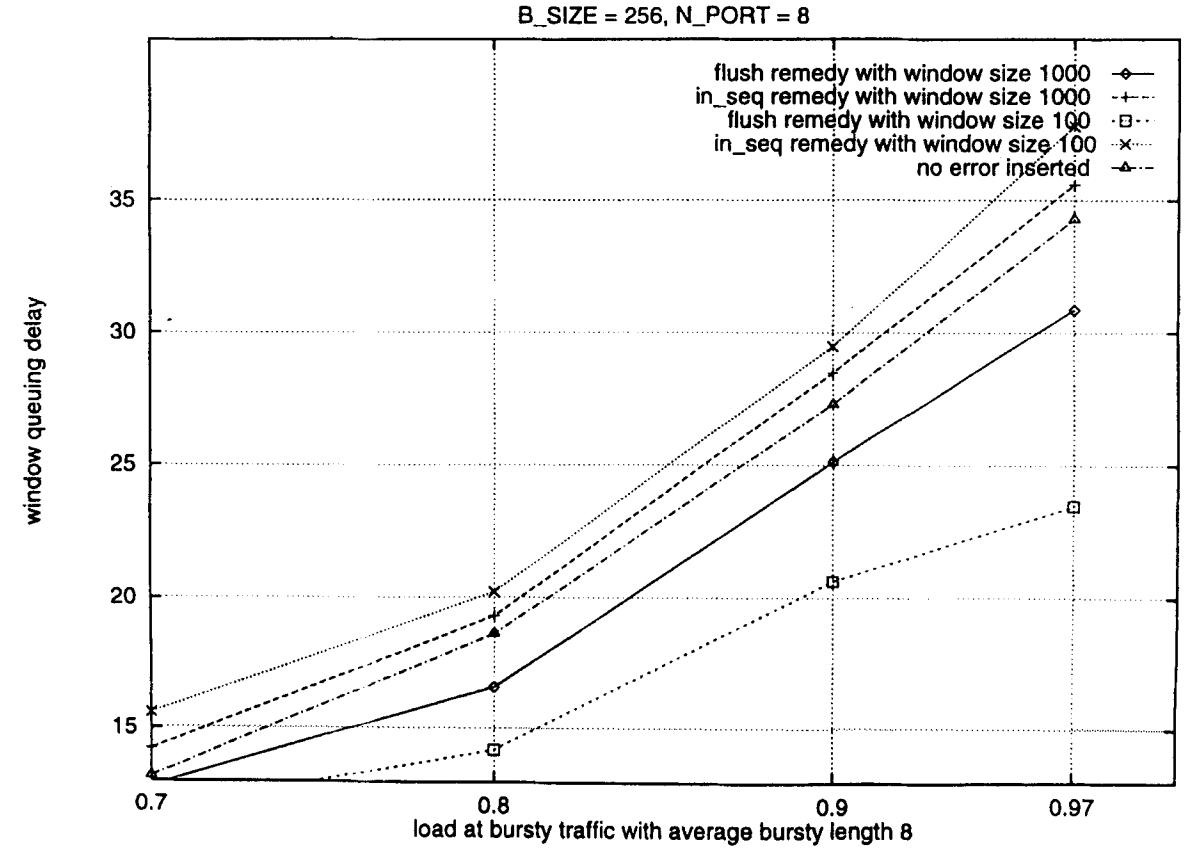

Figure 14. Queuing delay versus load at bursty traffic by changing window size

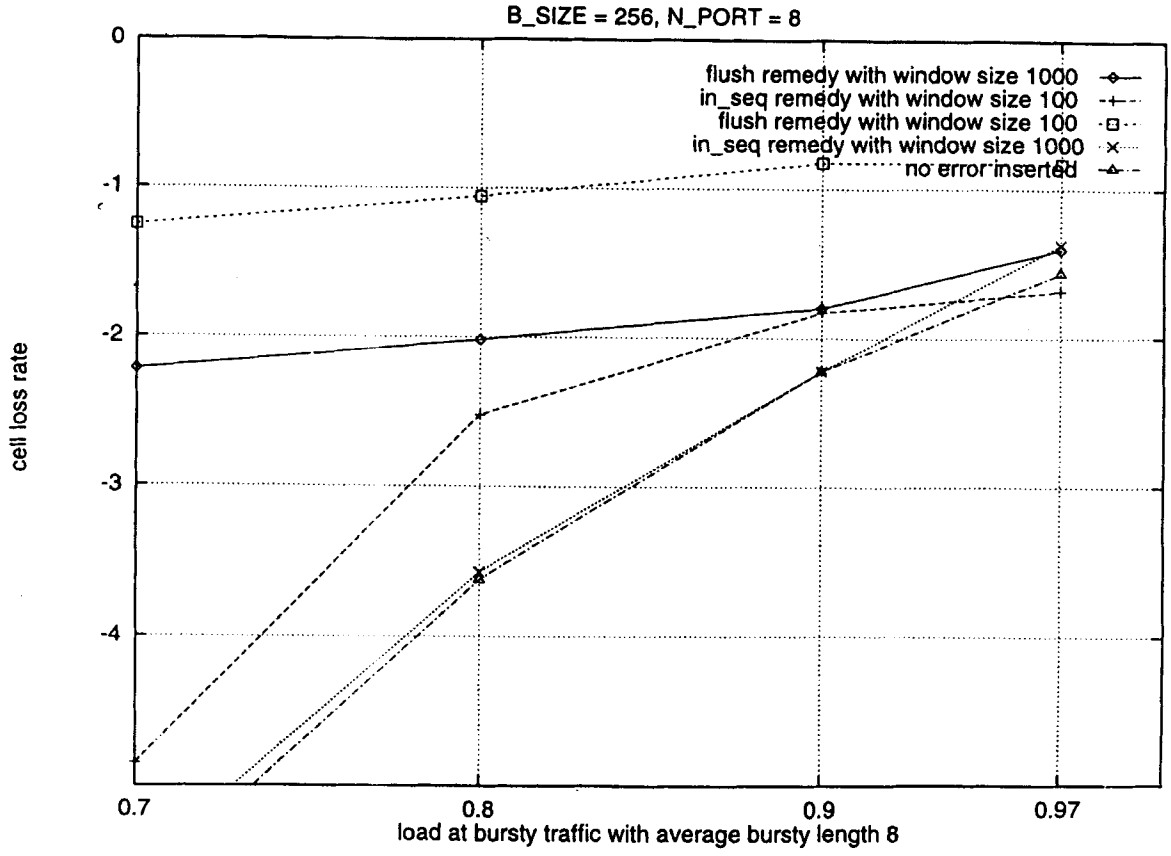

Figure 15. Cell loss rate versus load at bursty traffic by changing window size

under different error injection rates. We observed that under random uniform traffic we cannot easily tell the difference between these two schemes. The difference is enlarged under bursty traffic. If the error injection rate is small, we prefer the *In-Seq* scheme which has a negligible delay penalty, better cell loss rate performance and endures shorter time to converge to the error free situation.

4. CONCLUSION

Rapid evolution in the field of telecommunications has led to the emergence of new switching technologies to support a variety of communication services with a wide range of transmission rates in a common, unified integrated services network. At the same time, the progress in the field of VLSI technology has brought up new design principles of high-performance, high-capacity switching fabrics to be used in the integrated networks of the future. Among most proposals for switching architecture, the shared buffer memory switch (SBMS) is one of the best alternatives for implementation specially based on its reduced usage of buffer size at bursty traffic.

However, if the address chain memory in an SBMS is wrong, it will cause the port reading out cells not belonging to the proper port, and all cells destined to the port will never be read out again. More seriously, *fault spread* can occur, which renders all other ports in the same switch to be faulty in a short time. The phenomenon is mainly due to the central control processing rather than distributed control in the multistage switches.

Although there are some papers discussing the architecture of fault-tolerant switches, by and large, they were based on the Batcher–banyan switch. In the paper we have surveyed the methods for fault-tolerant SBMS architectures. Two schemes, *Flush* and *In-Seq* of fault tolerance have been proposed and comparisons have been made by computer simulation. The *Flush* scheme flushes all cells in the port after faulty location, and collects the addresses of the flushed cells for later usage. Therefore intuitively this scheme will increase the cell loss rate. On the other hand, the *In-Seq* scheme tries to output the cells after faulty location rather than flushing, and preserves the original cell sequence. However, this will introduce extra queuing delay for fixing the queue. From simulation we found that under lower error injection rate the *In-Seq* scheme suffers only a slight degradation in queuing delay compared with that of no error injected, and is faster to leave fluctuating situation after error injection. Therefore, we feel that the *In-Seq* scheme is preferred for its superior cell loss rate performance.

ACKNOWLEDGEMENT

This work was sponsored by the Telecommunication Laboratories (TL Y83015) and the National Science Council of Taiwan (NSC 81-0404-E009-135).

REFERENCES

1. F. Tobagi, 'Fast packet switch architectures for broadband integrated services digital networks', *Proc. IEEE*, January 1990, pp. 133–167.
2. T. Kozaki, N. Endo, Y. Matsubara, M. Mizukami and K. Asano, '32 × 32 shared buffer type ATM switch VLSI's for B-ISDN', *IEEE J. Selected Area in Communications*, **9**, (6), 1239–1247 (1991).
3. N. Endo, T. Kazaki, T. Ohuchi, H. Kuwahara and S. Gohara, 'Shared buffer memory switch for an ATM exchange', *IEEE Trans. Communications*, **41**, (1), 237–245 (1993).
4. H. Kuwahara, N. Endo, M. Ogino, T. Kozaki, Y. Sakurai and S. Gohara, 'A shared buffer memory switch for an ATM exchange', *Proc. ICC*, 1989, pp. 118–122.
5. S. Liew, 'Performance of input-buffered and output-buffered ATM switches under bursty traffic: simulation study', *Proc. Global Telecommunications Conference*, 1990, pp. 1919–1925.
6. D. Taylor *et al.*, 'Redundancy in data structures: improving software fault tolerance', *IEEE Trans. Software Engineering*, **6**, (6), 585–594 (1980).
7. D. Taylor *et al.* 'Redundancy in data structures: some theoretical results', *IEEE Trans. Software Engineering*, **6**, (6), 595–607 (1980).

Authors' biographies:

Yeong-Fong Lin was born in Chia-Yi, Taiwan, ROC on 10 June, 1969. He recieved the B.S. in Electrical Engineering from National Tsing Hua University, Hsinchu, Taiwan, ROC, in June 1992, and the M.S. in Electronics Engineering from the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, ROC, in 1994. He was involved in the research on ATM switch architectures. His M.S. thesis is entitled 'The fault-tolerant architectures of shared buffer memory switch'. Currently he is serving in the ROTC program in Taipei, Taiwan.

C. Bernard Shung received his B.S. in Electrical Engineering from National Taiwan University in 1981, and M.S. and Ph.D. in Electrical Engineering and Computer Science from University of California, Berkeley, in 1985 and 1988, respectively. He is currently an Associate Professor in the Department of Electronics Engineering, National Chiao Tung University in Hsinchu, Taiwan, ROC, where he joined the faculty in 1990.

He was a Visiting Scientist at the IBM Research Division, Almaden Research Center in San Jose, California, in 1988–1990. During the academic year of 1994–1995, he visited Qualcomm Inc. in San Diego, California. He was involved in architecture and circuit designs of communications and signal processing applications, including storage channel, pattern recognition, and wireless communications.

Dr. Shung's research interests include architecture and integrated circuit design for communications, signal processing, and ATM switches, and computer-aided design for integrated circuits and field programmable gate arrays. He has published more than 30 technical papers in related areas.