# DAMQ Self-Compacting Buffer Schemes for Systems with Network-On-Chip

Jin Liu and José G. Delgado-Frias

School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164-2752

***Abstract*****—***In this paper we present two novel buffer schemes for systems-on-chip applications that have an interconnection network. The proposed schemes are based on a DAMQ self-compacting buffer. These schemes outperform existing approaches. In addition the schemes have similar performance using only half of the buffer size used in other implementations. One of our schemes (DAMQ$_{min}$) provides an excellent technique to optimize buffer management providing a good throughput when the network has large load.*

*Keywords: Network on chip, systems-on-chip, DAMQ buffer, interconnection network.*

## 1. INTRODUCTION

As VLSI technology allows greater integration, system-on-chip (SoC) designs are being investigated in greater depth. By the end of this decade SoC will have up to four billion transistors [1]. SoCs are anticipated to have a number of components (or modules) that will interact to compute a solution. These components will need to communicate to pass data and/or control information. Having dedicated connections between any given module could be extremely complex as the number of modules increases. A viable alternative could be to use an interconnection network within the chip.

Using an interconnection network on chip needs be restricted in terms of area due to the constraints of being in a single chip. Thus, it is extremely important to use schemes that require less hardware resources and provide a good performance. In this paper we present two schemes that are based on a dynamically allocated multiple queue (DAMQ) buffer. The schemes provide similar performance as other statically allocated multiple-queue buffers using less hardware and, therefore, requiring less hardware.

This paper has been organized as follows. In Section II, the proposed DAMQ buffer schemes are described in detail. Performance evaluation and comparison results are reported in Section III. Some concluding remarks are included in Section IV.

## 2. NOVEL BUFFER SCHEMES

In this section we give a brief review of existing DAMQ buffer schemes then describe our two novel buffer schemes. These new schemes are DAMQ buffers with reserved space for all virtual channels (DAMQ$_{all}$) and DAMQ buffers with minimum reserved space for virtual channels (DAMQ$_{min}$).

### 2.1. Existing DAMQ buffer schemes

*1) Linked list buffer scheme*. In order to let multiple queues of packets share a DAMQ buffer, linked lists can be used to implement the buffer scheme [2,3]. The basic idea of this approach is to maintain ($k$+1) linked lists in each buffer: one list of packets for each one of the ($k$-1) output ports, one list of packets for the end node interface and one list of free buffer blocks. Corresponding to each linked list there is a head register and a tail register. The head register points to the first block in the queue and the tail register points to the last block. In each output queue, next block information also must be stored in each buffer block to maintain the FIFO ordering.

*2) Self-compacting buffer scheme*. To reduce hardware complexity of the linked list scheme, an efficient DAMQ buffer design self-compacting buffer was brought by [4]. The idea for this buffer scheme is to divide the buffer dynamically into regions with every region containing the data associated with a single output channel. If two channels are denoted as $i$, $j$ with $i<j$, then the addresses of buffer regions for the two channels $A_i$ $A_j$ will be $A_i<A_j$. There is no reserved space dedicated for any channel. Data is stored in a FIFO manner within the region for each channel. When an insertion of the packet requires space in the middle of the buffer, the required space will be created by moving down all the data which reside below the insertion

address. Similarly, when a reading operation conducted from the top of a region, data removed from the buffer may result in empty space in the middle of the buffer, then the data below the read address is shifted up to fill the empty space.

## 2.2. DAMQ with reserved space for all virtual channels

DAMQ dynamically allocate buffer blocks according to the packet received. Compared with statically allocated buffer scheme, the advantage of DAMQ is its efficient use of buffer space by applying free space to any incoming packet regardless its destination output port. Since there is no reserved space dedicated to each output channel, the packets destined to one specific output port may occupy the whole buffer space thus the packets destined to other output ports have no chance to get into the buffer. This is the case especially for small and compact routers with limited buffer space where wormhole switching technique and virtual channel mechanism are used. When several virtual channels multiplexed across the physical channel and share a buffer, the virtual channels which have packets accepted in the buffer prior to other virtual channels may hold the whole buffer space when the output port to next hop it destines to is busy. In order to overcome this shortcoming of DAMQ buffer schemes, we implement a new buffer organization scheme, DAMQ with reserved space for all virtual channels ($DAMQ_{all}$). $DAMQ_{all}$ is based on Self-compacting buffer (SCB) scheme.

It inherits most features of the SCB, the virtual channels of one direction of a physical channel share a buffer. The new feature added is that there is reserved space dedicated for each virtual channel, therefore at any time there is free space for the packets of "late" virtual channels which has not received packet and one virtual channel can never consume the whole buffer. As shown in Fig 1, two buffer slots are reserved for each virtual channel before the buffer accepts any incoming flit.

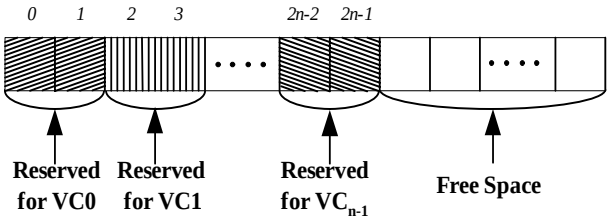


Fig. 1. $DAMQ_{all}$ Buffer space at the initial state.

The reserved spaces for each virtual channel are arranged sequentially according to the sequence numbers of the virtual channels. One register is used to point to the head of each reserved space, i.e. the head of the buffer region for each virtual channel. If two channels are denoted as $V_i$, $V_j$ with $i + 1 = j$, then the reserved region for $V_j$ will be placed right after the reserved region for $V_i$. The size of reserved space for each virtual channel can be adjusted, we choose two flits is because in our simulation experiments, two reserved flits scheme yields satisfactory performance while keeps more free space for sharing. When there is an incoming flit to the buffer, the $DAMQ_{all}$ operates as follows:

```
if (first flit for current VC){
   put it into buffer;
   increment counters;
}
else if
   (current VC doesn't fill reserved space for it) or
   (there is free slot left in buffer){
      if (last flit of current VC is next to first slot of
         next VC buffer space) {
         shift down all the flits and reserved space of
         the lower virtual channels one slot;
         increment head pointer for lower VCs;
         put flit into buffer;
         increment counters;
      }
      put flit into buffer;
      increment counters;
   }
```

When a flit is leaving the buffer, the $DAMQ_{all}$ operates as follows:

```
if (last flit in current VC){
   write flit to output port;
   decrement counters;
}
else{
   write flit to output port;
   decrement counters;
   shift up remaining flits of current VC;
   if (number of remaining flits of current VC >=
      reserved space number)
      shift up all the flits and reserved space of
      following VC one slot;
      decrement head pointer for following VC;
   }
}
```

As shown in Fig 2, the reserved space for each virtual channel is always kept if there is no flit or only one flit in the buffer region for a specific virtual channel.

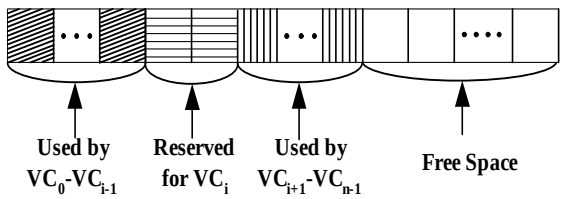


Fig. 2. $DAMQ_{all}$ Buffer space status in operations.

When the buffer performs shift up or shift down operations, the reserved spaces are treated same as the slots which are holding flits. Thus the order of the buffer space for virtual channels is kept conforming to the sequence of virtual channels. And once the number of current flits in buffer plus the number of reserved slots equals to the total amount of buffer slots, no more flit will be accepted unless this flit belongs to a virtual channel which has any reserved space available. Therefore, one or more virtual channels which have the flits come into the buffer at earlier time can never deprive the chance for other virtual channels which get flits later than them to get buffer. Moreover, once the earlier coming packets are blocked in the buffer, since there is still reserved space for other virtual channels, the network traffic will keep flowing, so the performance of the switch is also enhanced. This is the key improvement of $DAMQ_{all}$ scheme over SCB scheme.

## 2.3. DAMQ with minimum reserved space for virtual channels

$DAMQ_{all}$ improved SCB by reserving buffer space for each virtual channel to avoid the situation that a few virtual channels consume the whole buffer then other virtual channels can not get buffer even when those virtual channels which get buffer are blocked. In the simulation experiments, we found that $DAMQ_{all}$ is not the most efficient way to reserve space for virtual channels. In a specific time interval, there may be no packets destined for some virtual channels. Even worse situation is that there may be no packets destined for some virtual channels for a very long time. In either case, the reserved spaces for these idle virtual channels are wasted. In order to reserve the buffer space more efficiently and provide more space for flowing traffic, we implement another buffer organization scheme, DAMQ with minimum reserved space for all virtual channels ($DAMQ_{min}$). $DAMQ_{min}$ is also based on SCB scheme and the virtual channels of one direction of a physical channel still share a buffer. And based on the simulation results, we still set the number of reserved buffer space to two slots. The difference to $DAMQ_{all}$ is that at any time there is at most one reserved space for all the virtual channels. And if every virtual channel have flit present in the buffer, no reserved space will be kept in the buffer anymore. Thus $DAMQ_{min}$ use minimum space for reserve purpose. As shown in Fig 3, two buffer slots are reserved for the virtual channel which may firstly claim for buffer.

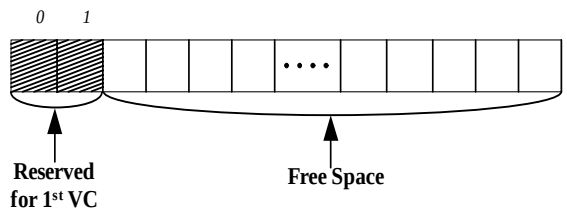


Fig. 3. $DAMQ_{min}$ Buffer space status in operations.

When there is an incoming flit to the buffer, the $DAMQ_{min}$ operates as follows:

```
if (first flit for current VC) and (there is a reserved
    space){
  put it into buffer;
  increment counters;
  set head pointer for current VC;
  if ( have enough free space in buffer ) and (not
     every VC are present)
     set the reserved space pointer to the slot next
     to current VC reserved space;
}
else if
  (current VC doesn't fill reserved space for it) or
  (there is free slot left in buffer){
     if (last flit of current VC is next to first slot of
        next VC buffer space) {
        shift down all the flits and reserved space of
        the lower virtual channels one slot;
        increment head pointers for lower VCs;
        put flit into buffer;
        increment counters;
        return;
     }
     put flit into buffer;
     increment counters;
  }
```

When a flit is leaving the buffer, the $DAMQ_{min}$ operates as follows:

```
if (last flit in current VC){
   write flit to output port;
   decrement counters;
   reset head pointer for current VC;
}
else{
   write flit to output port;
   decrement counters;
   shift up remaining flits of current VC;
   if (number of remaining flits of current VC >=
        reserved space number)
      shift up all the flits and reserved space of
      following VC one slot;
      decrement head pointer for following VC;
   }
}
if ( no reserved space is present in buffer ) and (not
   every VC are present){
   if (there are enough free buffer slots)
      set the reserved space pointer to the slot next to
        last VC reserved space;
}
```

Once a virtual channel has one flit come into the buffer, it will occupy two buffer slots which were reserved space before it comes in, thus there is actually still one slot reserved for it. As shown in Fig 4, this virtual channel will hold at least these two buffer slots unless it has no flit left in the buffer any more. Once every flits of a virtual channel moves out the buffer, the header pointer of this virtual channel will be reset to empty and there are no more buffer slots belong to it. Another two slots reserved buffer region may be created if possible. Reserved space is always placed right after the buffer region of virtual channel which is the latest one to have flit into the buffer. When the buffer performs shift up or shift down operations, all reserved slots are treated same as the slots which are holding flits.

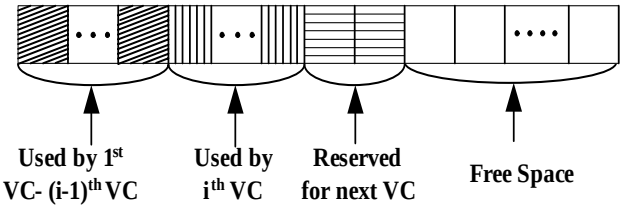


Fig. 4. DAMQ$_{min}$ Buffer space status in operations.

By dynamically creating reserved space for virtual channels, DAMQ$_{min}$ presents a very efficient way to use buffer space, there is always minimum buffer space used for reserve purpose, so there are more free space available for flowing traffic.

# 3. PERFORMANCE EVALUATION

In this section, we describe the results of simulation experiments carried out to evaluate the performance of the proposed buffer organization schemes. Firstly we describe our methodology and configuration of simulation environment. Then we examine the performance of DAMQ$_{all}$ and DAMQ$_{min}$ in greater detail.

## 3.1. Simulation experiments

We have carried out our simulations using flexsim1.2 [6] which is a simulator for flit-level simulation of torus/mesh networks. The architecture we simulate is a 4-ary 2-cube message exchanging system with wrapped around channels as shown in Fig 5.

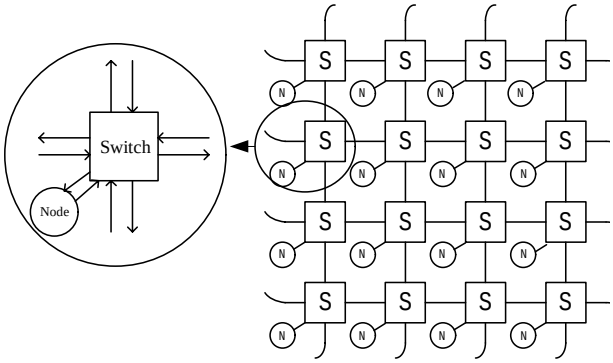


Fig. 5. The base system for our simulation

Each switch is attached to one end-node which has one injection channel to the switch and one input channel to receive message from network. Physical channels are bidirectional duplex channel. Network traffic is uniformly distributed. Every end node generates packets with randomly determined destination and injects them into the network. Other pertinent simulation configuration parameters are listed as follows:

- Routing flit delay is set to 1 cycle.
- Data flit delay is set to 1 cycle.
- Buffers at local end nodes are infinite.
- Packets size is set to 32 flits.
- Switching technique used is wormhole.
- Routing protocol used is E-cube.

In our experiments, we vary the applied load, buffer size and virtual link number; study their impact on throughput and message latency. Since messages are divided to flits when transmitting, to increase message length has the similar effect on increasing traffic load as to shorten the average injection period for each node.

We set the message length at 32 flits and make the network into saturation state by shortening injection period for each node. Traffic load rate is derived by formula: (load rate) x (number of channels) = (number of nodes) x (message length / average injection period) x (average routing distance). [6] The throughput is defined as the number of flits received per node per cycle. The message latency is measured as the average time span for every packet between the moment it generated and the reception of the whole packet at destination.

## 3.2. Effect of number of virtual channels and buffer size

To increase the number of virtual channels multiplexing a physical channel can improve switch performance, but too many virtual channels will not only incur expensive hardware expense but also increase the message delay. In our simulation experiments, we set virtual channel number of one direction of a physical channel to 4 and 8, thus the advantages brought by virtual channel will not be overshadowed by its shortcoming. We evaluate the performance of the switch in terms of the throughput and the message latency which are two most important performance metrics. We compare our buffer scheme $DAMQ_{all}$ and $DAMQ_{min}$ to the traditional statically allocated buffer scheme (SAMQ) used for virtual channels which is reported in [5].

*1) Simulation results on four virtual channels:* In this part we present the simulation results obtained when 4 virtual channels are multiplexing a physical channel. We examine the performance for three different size buffers on SAMQ and two different size buffers on $DAMQ_{all}$ and $DAMQ_{min}$, The throughput and message latency are shown in Tables 1 and 2, respectively. It is well known that there is no significant difference for different buffer scheme while the network is not saturated. In our simulation experiments, we keep increasing the traffic load so that we can compare the performance of different buffer schemes when the network is in saturation status. As shown in Fig 6, along with the network saturation process, our $DAMQ_{all}$ and $DAMQ_{min}$ have higher throughputs than SAMQ if they all use same size buffer. $DAMQ_{all}$ achieves approximate the same maximum throughput as SAMQ. However it only uses half of the buffer space used by SAMQ. $DAMQ_{min}$ gets even higher maximum throughput than $DAMQ_{all}$, because the former saves more buffer slots ready to be used than the latter. Furthermore, at the same maximum throughput level, both $DAMQ_{all}$ and $DAMQ_{min}$ have less latency than SAMQ as shown in Fig 7.

TABLE I
THE PERFORMANCE 4-ARY, 2-CUBE NETWORK COMPOSED OF BLOCK SWITCHES. SHOWN IS THE THROUPUT OBTAINED FROM SIMULATIONS WHERE 4 VIRTUAL CHANNELS USED.

| Buffer Type | BS per VC | Throughput Versus Applied Traffic Load Rate | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0.02 | 0.06 | 0.10 | 0.14 | 0.18 | 0.55 | 1 |
| SAMQ | 4 | 0.16 | 0.31 | 0.57 | 0.62 | 0.65 | 0.66 | 0.66 |
| | 8 | 0.15 | 0.32 | 0.58 | 0.64 | 0.68 | 0.69 | 0.69 |
| | 16 | 0.15 | 0.32 | 0.58 | 0.66 | 0.69 | 0.71 | 0.71 |
| $DAMQ_{all}$ | 4 | 0.16 | 0.31 | 0.57 | 0.63 | 0.67 | 0.68 | 0.68 |
| | 8 | 0.16 | 0.31 | 0.58 | 0.65 | 0.70 | 0.71 | 0.71 |
| $DAMQ_{min}$ | 4 | 0.16 | 0.32 | 0.58 | 0.64 | 0.68 | 0.69 | 0.69 |
| | 8 | 0.16 | 0.31 | 0.59 | 0.66 | 0.70 | 0.72 | 0.72 |

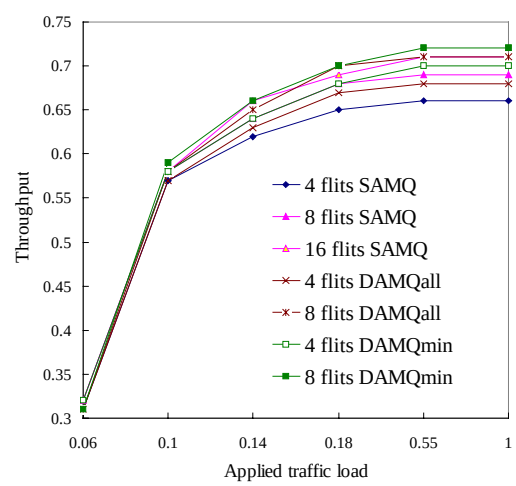


Fig. 6. Comparison on throughput between 4/ 8 flit-buffer DAMQall / DAMQmin and 4/ 8/ 16 flit-buffer SAMQ

TABLE II
THE PERFORMANCE 4-ARY, 2-CUBE NETWORK COMPOSED OF BLOCK SWITCHES. SHOWN IS THE MESSAGE LATENCY OBTAINED FROM SIMULATIONS WHERE 4 VIRTUAL CHANNELS USED.

| Buffer Type | BS per VC | Message Latency Versus Applied Traffic Load Rate | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0.02 | 0.06 | 0.10 | 0.14 | 0.18 | 0.55 | 1 |
| SAMQ | 4 | 54 | 73 | 119 | 144 | 171 | 207 | 217 |
| | 8 | 53 | 71 | 119 | 150 | 173 | 217 | 226 |
| | 16 | 53 | 70 | 120 | 155 | 207 | 256 | 273 |
| $DAMQ_{all}$ | 4 | 55 | 71 | 116 | 144 | 173 | 209 | 219 |
| | 8 | 54 | 71 | 119 | 154 | 174 | 242 | 247 |
| $DAMQ_{min}$ | 4 | 53 | 70 | 120 | 142 | 178 | 210 | 218 |
| | 8 | 54 | 69 | 118 | 156 | 192 | 237 | 249 |

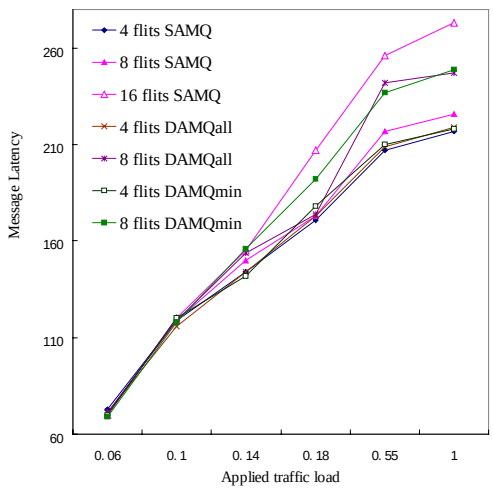


Fig. 7. Comparison on Latency between 4/ 8 flit-buffer DAMQall / DAMQmin and 4/ 8/ 16 flit-buffer SAMQ

2*) Simulation results on eight virtual channels:*

We present the simulation results obtained when 8 virtual channels are multiplexing a physical channel in this section. Because more virtual channels are used, the throughput and latency are higher than 4 virtual channels situation when the network is saturated. The throughput and message latency are shown in Table 1 and Table 2 respectively. $DAMQ_{all}$ and $DAMQ_{min}$ again get better performance over SAMQ. As shown in Fig 8, with only half size buffer, $DAMQ_{min}$ even get higher maximum throughput than SAMQ and $DAMQ_{all}$ get same throughput as SAMQ. The reason is when there are more virtual channels involved, $DAMQ_{min}$ and $DAMQ_{min}$ can have more free buffer space to use and $DAMQ_{min}$ make the most efficient usage on the buffer. Also as shown in Fig 9, there is no unwanted message latency introduced for $DAMQ_{all}$ and $DAMQ_{min}$, they have approximate same latency as SAMQ that are using double size buffer.

TABLE III

THE PERFORMANCE 4-ARY, 2-CUBE NETWORK COMPOSED OF BLOCK SWITCHES. SHOWN IS THE THROUPUT OBTAINED FROM SIMULATION WHERE 8 VIRTUAL CHANNELS USED..

| Buffer Type | BS per VC | Throughput Versus Applied Traffic Load Rate | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0.02 | 0.06 | 0.10 | 0.14 | 0.18 | 0.55 | 1 |
| SAMQ | 8 | 0.16 | 0.32 | 0.63 | 0.74 | 0.80 | 0.80 | 0.79 |
| | 16 | 0.16 | 0.32 | 0.63 | 0.75 | 0.81 | 0.82 | 0.81 |
| $DAMQ_{all}$ | 8 | 0.16 | 0.32 | 0.64 | 0.77 | 0.81 | 0.82 | 0.81 |
| $DAMQ_{min}$ | 8 | 0.16 | 0.30 | 0.64 | 0.77 | 0.82 | 0.83 | 0.83 |

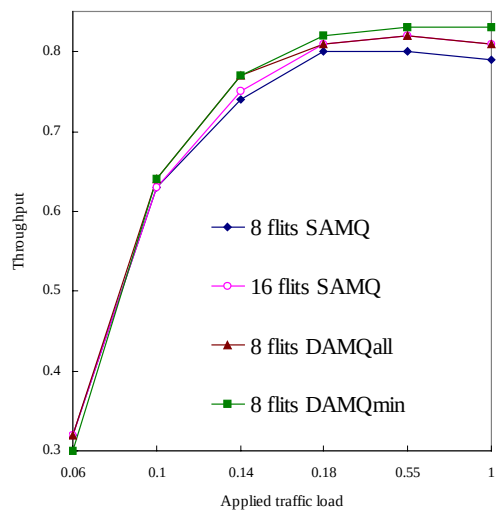


Fig. 8. Comparison on Throughput between 8 flit-buffer DAMQall / DAMQmin and 8/ 16 flit-buffer SAMQ

TABLE IV

THE PERFORMANCE 4-ARY, 2-CUBE NETWORK COMPOSED OF BLOCK SWITCHES. SHOWN IS THE MESSAGE LATENCY OBTAINED FROM SIMULATIONS WHERE 8 VIRTUAL CHANNELS USED.

| Buffer Type | BS per VC | Message Latency Versus Applied Traffic Load Rate | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 0.02 | 0.06 | 0.10 | 0.14 | 0.18 | 0.55 | 1 |
| SAMQ | 8 | 55 | 74 | 147 | 214 | 308 | 375 | 394 |
| | 16 | 54 | 73 | 150 | 213 | 347 | 439 | 455 |
| $DAMQ_{all}$ | 8 | 55 | 73 | 147 | 219 | 357 | 438 | 443 |
| $DAMQ_{min}$ | 8 | 54 | 73 | 147 | 218 | 357 | 439 | 458 |

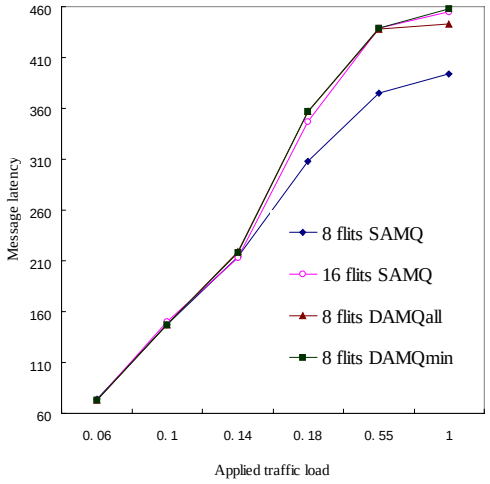


Fig. 9. Comparison on Latency between 8 flit-buffer DAMQall / DAMQmin and 8/ 16 flit-buffer SAMQ

## 4. CONCLUDING REMARKS

In this paper we have presented two novel buffer schemes based on a DAMQ self-compacting buffer. These schemes outperform existing approaches. In addition the proposed schemes have similar performance using only half of the buffer size used in SAMQ. $DAMQ_{min}$ provides an excellent scheme to optimize buffer management providing a good throughput when the network has a larger load. Implementing the proposed schemes in hardware would require minor modifications to early implementations of the self-compacting buffer [7].

## 5. REFERENCES


[1] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, Vol. 35 , No. 1, pp. 70-78, Jan. 2002

[2] R. Sivaram, C. B. Stunkel, and D. K. Panda, "HIPIQS: A High Performance switch architecture using input queueing," *IPPS/SPDP '98*, pp. 134–143, Orlando, FL, March 1998.

[3] Y. Tamir and G. L. Frazier, "Dynamically-allocated multiqueue buffers for VLSI communication switches," *IEEE Transactions on Computers*, vol. 41, no. 2, pp. 725–737, June 1992.

[4] J. Park, B. O'Krafka, S. Vassiliadis, and J. Delgado-Frias, "Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers," *IEEE Supercomputing '94, The Conference on High Performance Computing and Communications*, pp. 713–722,Nov.14-18 1994

[5] W. Dally. "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[6] S. Warnakulasuriya and T.M. Pinkston, "Characterization of Deadlocks in k-ary n-cube Networks," IEEE Trans. Parallel and Distributed Systems, vol. 10, no 9, pp. 904-932, Sept. 1999.

[7] J. G. Delgado-Frias and R. Diaz, "A VLSI Self-Compacting Buffer for DAMQ Communication Switches," *IEEE Eighth Great Lakes Symposium on VLSI*, pp. 128-133, Lafayette, Louisiana, February 1998.