Flow Control for Interconnection Networks

Stergios Stergiou

***summary*—An interconnection network is characterized by its topology, routing and flow control. The topology is a directed graph, where the nodes represent the routing modules and the edges represent the channels. Routing specifies how a packet chooses a path on this graph. Flow control examines the allocation of resources, mainly channels and buffers to a packet as it traverses this path. This paper summarizes five works on flow control, each optimizing different performance and reliability metrics, such as throughput, latency and quality of service, in an attempt to examine them under the prism of their applicability for on-chip interconnects. A Critical review and a number of optimizations are also presented.**

I. Introduction

Over the last years, significant effort has been devoted into optimizing network performance of multi-computers. Recently [1], network constructs have been proposed as an alternative for replacing on-chip interconnection networks that are traditionally bus-based.

Designing a flow control (FC) mechanism involves tradeoffs between latency, throughput and implementation complexity. Classic FC approaches comprise Circuit Switching, Store and Forward, Virtual Cut Through and Wormhole (for a reference, see [2].)

Currently, wormhole routing, a routing mechanism for assigning resources on sub-packet data units designed for low latency and high throughput at a low cost, is widely accepted and is more often than not adopted for novel proposed flow control techniques.

Virtual-Channel Flow Control has been proposed as a throughput enhancing methodology in order to alleviate idling of channels due to resource coupling [3].

For the case where channels span distances across single clock cycle boundaries, elastic interconnects have been proposed as a flow control technique that decreases both latency and buffering requirements [4].

Where packet prioritization is desirable, a FC technique that preempts network resources in favor of higher priority packets has been proposed [5].

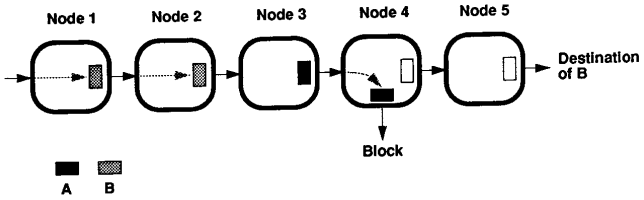

Fig. 1. B is blocked behind A while all physical channels remain idle.

This paper was created for the requirements of Stanford's EE382C, Interconnection Networks, Spring 2005, on May 1, 2005.

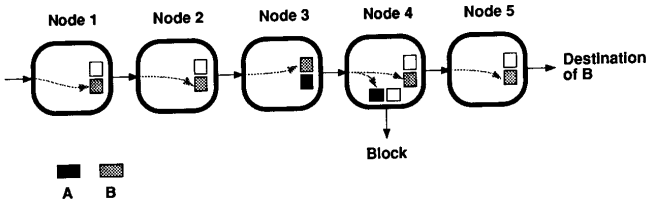

Fig. 2 Virtual Channels provide additional channels, allowing B to pass A.

II. Virtual-Channel Flow Control

A. Introduction

Interconnection networks comprise two resource types: Channels and Buffers. Typically, a single buffer is associated with a physical channel. Once a packet A is allocated a buffer b_i, on other packet B can utilize the corresponding channel c_i until A releases b_i. This scenario can lead to unnecessary channel underutilization, and is depicted in Fig. 1.

In order to alleviate this problem, virtual channels are proposed. A virtual channel comprises a flit buffer and the corresponding state information. It decouples buffer from channel allocation, by providing multiple buffers for each physical channel on the network. Fig. 2 illustrates the addition of virtual channels on a network.

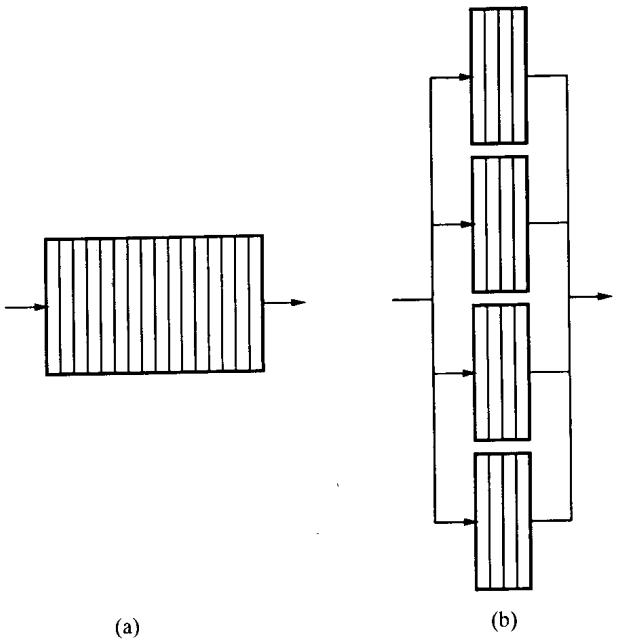

Fig. 3 (a) FIFO corresponding to single virtual channel input (b) four virtual channels buffer organization.

B. Functionality

A network that implements virtual channel flow control organizes its flit buffers as shown on Fig. 3. Buffers for each virtual channel are allocated independently of the others. This flexibility leads to higher overall channel utilization and therefore, higher throughput. Lanes can be assigned to different flows, hence even packets that span many hops do not necessarily cause blocking.

Flow control operation differs from the simple wormhole case, in the sense that it is performed at two levels. Virtual channel allocation is performed at a packet level, while physical channel bandwidth is performed at flit level. More concretely, virtual channels compete for physical channel bandwidth on a flit by flit basis.

Virtual channel state is stored both at the source and the target node of the physical channel of the corresponding VC. Source node maintains a status register that contains the state of the corresponding virtual channel. It holds a flag depicting whether the VC is free, a counter on the number of available flits at the VC buffer and optionally, a priority. Target node contains the actual VC buffer and another status register for the three possible states of the VC (free, waiting for an output, active.)

Once a packet is assigned a VC, transmission through an output physical channel is performed on a flit by flit basis. VCs are deallocated after the transmission of the packet's tail flit.

Let the number of VCs be l, the total number of flit buffers at the receiver b, and the number of priority bits pri. Then, the status storage S_{vc} required per physical channel is equal to:

$$S_{vc} = l(\lg(\tfrac{b}{l}) + pri + 1) + l(2\lg(\tfrac{b}{l}) + 2)$$

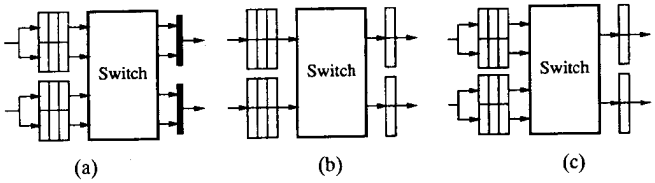

Fig. 4 Integrating VCs with switches. (a) replacing FIFOs with multilane buffers, (b) multiplexing the switch, (c) multiplexing only the outputs of the switch

C. Integration

There are three approaches to VC integration on switches. The first consists of replacing input FIFOs with multilane buffers. This approach, while conceptually simple, imposes unnecessary high requirements to the switch: As VCs increase, the number of inputs that the switch needs to handle increase as well. Moreover, flow control must be augmented to identify lanes. This approach is depicted in Fig. 4 (a).

By observing that the average data rate out of a set of VCs logically linked to a physical channel is limited by the channel bandwidth, we note that a single switch input is sufficient for each physical channel. A small output buffer decouples switch allocation from physical channel allocation. VCs are then multiplexed onto the single switch input corresponding to the physical input channel. Arbitration is more involved however. This approach is depicted in Fig. 4 (b).

An intermediate solution that simplifies arbitration is to only multiplex the outputs of the switches. In this case, an input VC competes only for switch output ports, and not for input ports. This approach is depicted in Fig. 4 (c).

D. Observations

Virtual-Channel Flow Control has been a widely adopted idea since it provides solutions to many blocking and deadlock interconnection issues. Its applicability to the area of on-chip networks remains to be verified however. Issues that need to be examined comprise:

(i) *Area of the VC buffers.* Since on-chip networks are area-sensitive, small buffers are likely to be utilized. In this case, if the VCs are mapped to RAMs, the decoding circuit overhead is relatively large. Therefore, custom buffer architectures need to be explored.

(ii) *Fixing a constant number of VCs per physical channel.* If the routing performed is source-based, and the actual routing information is mostly static, then an analysis can be performed on the number of maximum paths that can overlap on a given physical channel. The corresponding VC count can then be custom tailored.

(iii) *General applicability of VC integration.* If the functionality of multiple VCs can be emulated by the addition of more inputs on a switch, it may be the case that the latter approach is preferable, since NoCs are not pin-limited. The impact on the critical path of the switch of adding more input ports, versus the corresponding impact of adding more VCs is a topic of research.

III. Elastic Interconnects

A. Description

It is known that global wire speeds do not relatively scale well with newer technologies [6]. Therefore, the interconnect delay can soon become a critical factor on the performance of chip-multiprocessors.

Elastic Interconnects is a proposed solution to flow control asynchronous channels whose length spans more than one clock cycle. They are based on the idea of "compressing" waveforms on a channel, which will become clear after the following description.

On Elastic Interconnects, (power) repeaters are replaced by three-state repeaters. These repeaters are controlled by a single flow control wire that traverses the channel in the opposite direction. An elastic channel is depicted in Fig. 5.

As the flow control wire traverses the channel backwards, the three-state repeaters hold the data currently on the channel at the corresponding position. The data that are fed to the current stage from the following stage going backwards, are lost, until the flow control line reaches the following stage and places it in Hi-Z.

Assuming that no transitions occurred on the waveform on the channel, this technique simply "compresses" the waveform as the flow control flows backwards. The reverse procedure applies when the flow control line is dropped, in which case, the waveform on the channel is "decompressed".

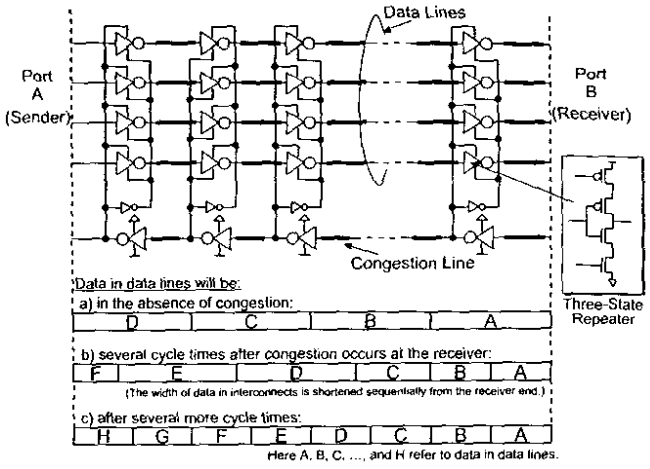

Fig. 5 Elastic Interconnect.

B. Observations

It is observed that, in order for elastic interconnects to operate normally, at least two three-state repeaters need to exist within the length covered by the channel during a single clock cycle, since otherwise, data would be lost. Moreover, in order to minimize the effect of three-stating a repeater at waveform transitions, more than two repeaters should be present per clock cycle.

The paper makes highly convincing arguments for the applicability of the approach, by integrating it into two relatively large test chips (~6M and ~63M transistor count.) The test chips however, operate at 100MHz and 200MHz.

The natural question then follows: Is the approach applicable for high-speed NoCs? An objection may be raised for the impact of clocking waveforms at transition points.

Let us try to eliminate the problem, by bringing the design into the synchronous world. Assume that a channel is N clock cycles long. Normally, pipeline registers would be placed at clock-cycle distances (see Fig. 6).

Let us instead latch the channel at half-clock cycle long distances, as depicted in Fig. 7. In order to replicate the reasoning of the elastic interconnects, each repeater is designed as depicted in Fig. 8.

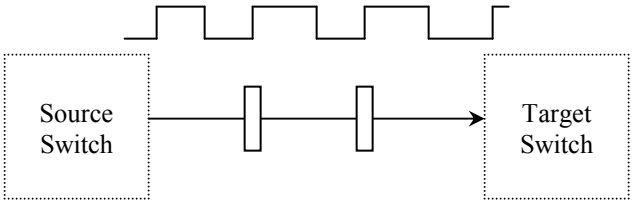

Fig. 6 Channel Pipelining.

It is noted that the cost of the control logic for the repeater is a single NAND gate and a flip-flop. The total number of repeaters on the channel is equal to 2N, for N-pipelined channels. By comparison, if credit flow control was utilized, at least 3N buffers would be necessary, 2N for the round-trip, and N for the pipeline registers on the channel.

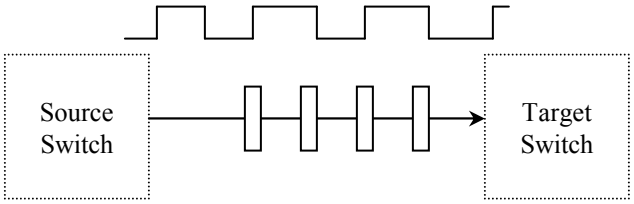

Fig. 7 Channel Pipelining at both clock edge transitions.

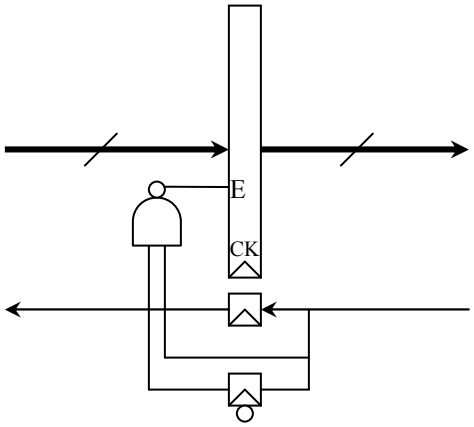

Fig. 8 Repeater Logic. The clock is implicit, and is inverted when the repeater is positioned at negative clock edges on the channel.

The approach of synchronous elastic interconnects has been implemented in SystemC and simulated under ModelSim. Fig. 9 shows the waveforms of a simple testbench.

We observe that this design is highly adapted to NoCs, since it assumes synchronous channels, an assumption that is difficult to satisfy for off-chip interconnects. Further optimizations can be performed. For instance, if idle or null type flits exist on the NoC, they need not be stalled on the channel. It seems very likely that this approach, combined with null flit on-the-fly removal will outperform Credit Flow Control in terms of latency, while utilizing only 66% of flit buffers.

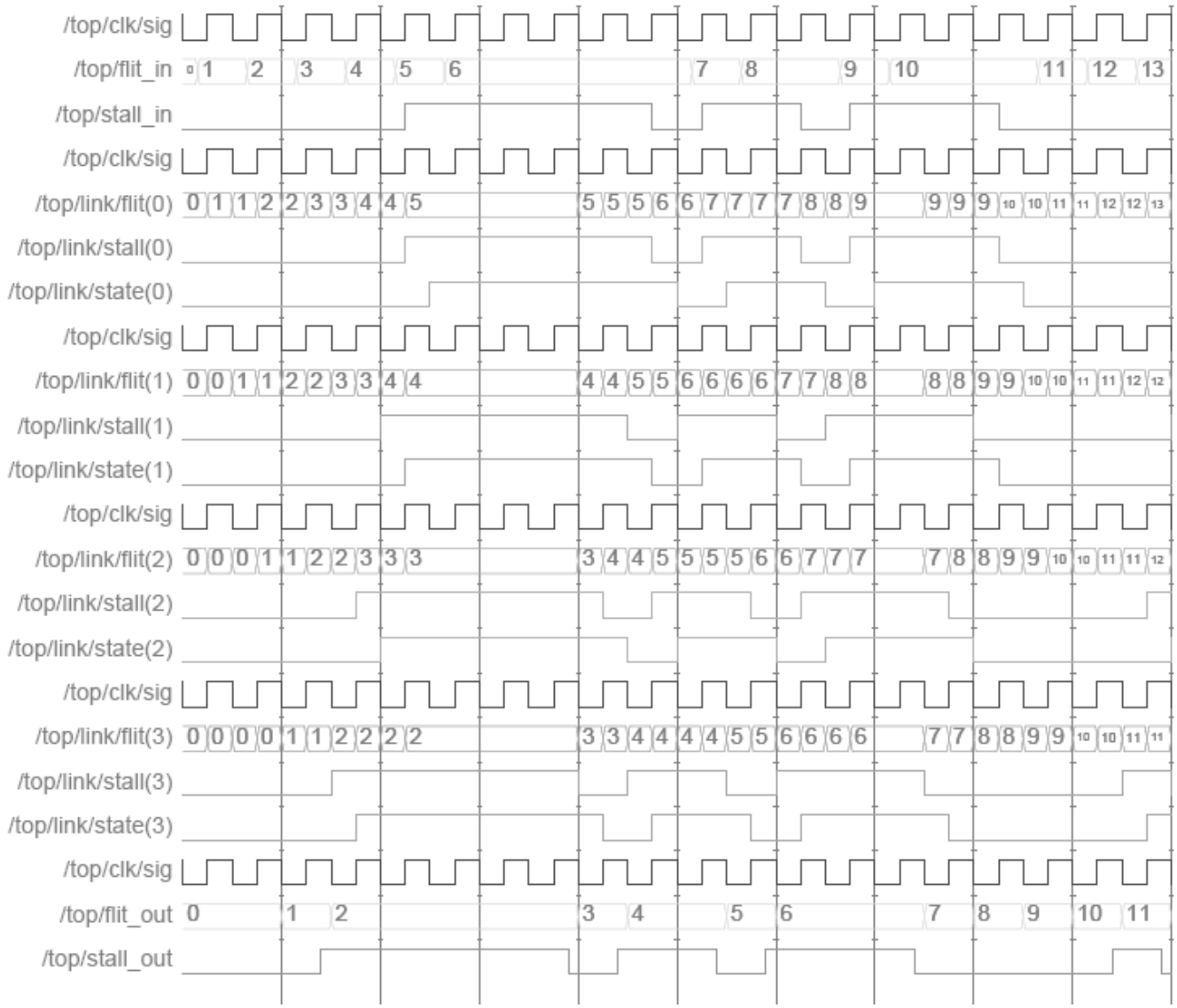

Fig. 9 Repeater Testbench. Flits are provided through /top/flit_in and are consumed at /top/flit_out. /top/stall_out models the flow control signal.

IV. Packet Prioritization

A. Introduction

In wormhole networks, priority inversion occurs when a high priority packet requests the channel held by a low priority packet. However, if a high priority packet preempts the channel, advances to the next node, and makes its own route there, the priority inversion can be avoided.

The flow control mechanism proposed to handle packet priorities is named Throttle and Preempt Flow Control, and consists of two fundamental policies: Throttle policy always reserves space in network resources for a high priority packet to preempt, and Preempt policy enables high priority packets to actually preempt network resources.

B. Throttle Policy

Suppose (see Fig. 10) when input buffer ib of node 2 is full with flits of packet P_q with low priority q, packet P_p with higher priority p arrives at node 3 and requests channel oc. Even though the priority of P_p is higher that the priority of P_q, packet P_p cannot preempt channel oc from P_q, since input buffer ib is full.

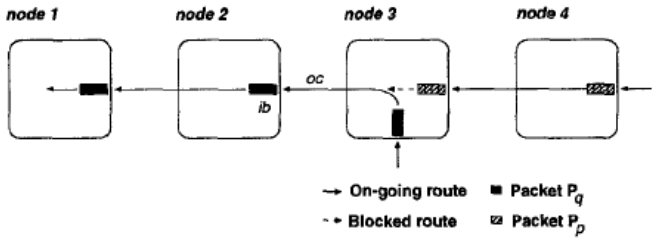

Fig. 10. Without throttle policy, a high priority packet P_p cannot preempt chanel oc.

Throttle policy has distinct treatment for packets with different priorities: a high threshold for a high priority packet, and a low threshold for a low priority packet. More concretely, a packet P is allowed to move to an input buffer, only if the buffer has room to receive more than $MAX_{PRI} - q - 1$ flits. Therefore, when packet P, requests the channel held by packet P, it can always preempt it, since there is room in the input buffer for at least $p - q$ flits.

This throttle policy enforces the size of an input buffer as MAX_{PRI} flits or more than, in order that even packets with the lowest priority could come into an input buffer. Fig. 11 illustrates that, for the input buffer of four flits and MAX_{PRI} of four, packet P_i with priority i can use the input buffer only up to i + 1 flits.

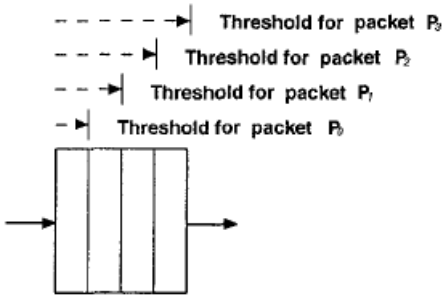

Fig. 11. Throttle policy.

C. Preempt Policy

Preempt policy lets a high priority packet preempt the network resources from a low priority packet. This preemption occurs over two adjacent nodes at a time, i.e., for the output channel at the previous node and for the associated input buffer at the next node. Preemption occurs over two nodes sequentially, first output channel preemption and then input buffer preemption.

1) Output Channel Preemption

Output channel oc held by packet P_q, with priority q is preempted by packet P_p, with priority p, if: (i) the header flit of packet P_p, arrives, (ii) there is no free output channel for packet P_p, (iii) packet P_p, desires the channel oc, (iv) $q < p$. Channel preemption is depicted in Fig. 12. Fig. 12 (a) shows that packet P_p enters node 3 and packet P_q is occupying output channel oc. Then, packet P_p interrupts packet P_q, and moves forward through output channel oc, as shown in Fig. 12 (b).

The remaining part of the packet P_q is blocked during the interruption and its transmission is resumed after the tail flit of packet P_p has left channel oc and its associated input buffer ib, as shown in Fig. 12 (c). Multiple preemptions may occur when another packet P_n with priority $n > p$ comes to node 3 and requests channel oc.

Subsequently, channel oc is again preempted by packet P_n, and packet P_p is blocked as well. After packet P_n has completely left channel oc and input buffer ib, the transmission of P_p is resumed. Moreover, after the complete departure of packet P_p, packet P_q can again be resumed.

In order to support the above mechanism, a preemption stack for each output channel is required. The preemption stack is a set of registers to record which input buffers hold preempted packets, in the ascending order of their priorities. The top register of the preemption stack points to the input buffer which holds the packet with the highest priority among preempted ones. On preemp-

tion, for the newly interrupted packet P_{top}, its input buffer's identifier is pushed onto the top of the preemption stack, since P_{top} has higher priority than any other packets already preempted.

When the output channel and its associated input buffer are freed and if there is no arrival of a higher priority packet than packet P_{top}, a request is granted from input buffer to hold P_{top}, and the top entry of the stack is popped off. The number of stack registers is $MAX_{PRI} - 1$.

2) Input Buffer Preemption

After preempting output channel oc, packet P_p preempts input buffer ib which is already held by packet P_q, as shown in Fig. 12 (b). Upon arrival of the header flit of packet P_p at a non-free buffer ib, input buffer ib is taken over for packet P_p. From this time, input buffer ib is fed with flits of packet P_p, and sends out these flits. Only after the tail flit of packet P_p has departed from input buffer ib, packet P_q can be again fed into, as shown in Fig. 12 (c).

During the interruption of packet P_q at node 2 and node 3, the downstream nodes continue to move forward the precedent flits of packet P_q. After all precedent flits pass through, reserved channels for packet P_q will be idle at the downstream nodes before its resumption. Such an idle on timing is inevitably inserted into the middle of packet P_q.

Two hardware constructs are necessary in order to support the above mechanism: (i) history stack and (ii) LPFO (Last Packet First Out) buffer. Recall that the header flit makes a routing decision and trailing data flits follow it in wormhole routing. Hence, in generic wormhole routing, a register is needed to keep the routing decision for each input buffer. However, for throttle and preempt flow control, before the tail flit of a packet passes though an input buffer, it may be preempted by another one. Therefore, a set of registers to store several routing decisions for each input buffer, the history stack, is required.

Additionally, the LPFO buffer makes flits of the last arrived packet be sent out first. Hence, flits of a high priority packet can bypass those of lower priority packets. After transmitting the tail flit of the last packet, the LPFO buffer resumes the transmission of the previous packet.

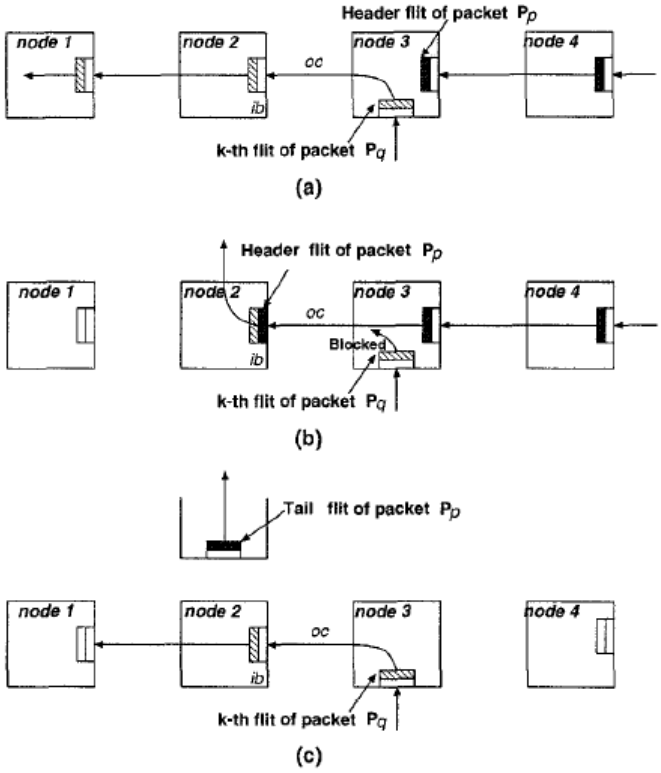

Fig. 12. Preempt policy.

D. Observations

While not explicitly mentioned in the paper, this work presents a clever way for "folding" the Virtual Channels of a specific physical channel onto a single buffer. The novelty seems to be the fact that if a high priority path occupies a specific empty buffer, then the buffering available to it will be greater than the corresponding one in the case of virtual channels.

We observe that this mechanism does not really provide priority groups, i.e. there cannot exist two paths with the same priority. Therefore, the proposed flow control comes at the expense of unfairness, since artificially lower priority connections will livelock

in the presence of artificially higher priority ones. This does not seem to be a big issue however, as it is known that locally unfair virtual channel arbitration for physical bandwidth implies lower latency.

One would need to examine the additional buffering requirements for the history and preemption stack. Impact on the critical path of the switch seems to be negligible. Moreover, since no data are stored on these buffers, the overhead is manageable. Additionally, the overall complexity of the mechanism seems to be low and seems to be especially applicable to on-chip interconnects.

REFERENCES

[1] William J. Dally and Brian Towles, *"Route packets, not wires: on-chip interconnection networks,"* in Proceedings of the Design Automation Conference, pp. 684–689, Las Vegas, NV, June 2001.

[2] William J. Dally and Brian Towles, "*Principles and Practices of Interconnection Networks*," ISBN: 0-12-200751-4, Morgan Kaufmann, 2003.

[3] W. J. Dally, *"Virtual-Channel Flow Control,"* IEEE Trans. Parallel Distrib. Syst. 3(2), 1992, pp 194—205.

[4] Mizuno, Masayuki, Dally, William J., and Onishi, Hideaki, *"Elastic Interconnects: Repeater-inserted Long Wiring Capable of Compressing and Decompressing Data,"* in Proceedings of the IEEE International Solid-State Circuits Conference, pp. 346-347, 464, 2001.

[5] H.Song, B.Kwon and H.Yoon, *"Throttle and Preempt: A Flow Control Policy for Real-Time Traffic in Wormhome Networks,"* Journal of Systems Architecture, 45(8), 1999.

[6] Ho, R., Mai, K.W., Horowitz, M.A., "*The Future of Wires,"* in Proceedings of the IEEE, pp 490—504, 89(4), Apr 2001.