Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2009-036 August 18, 2009

Guaranteed in-order packet delivery using Exclusive Dynamic Virtual Channel Allocation

Mieszko Lis, Keun Sup Shim, Myong Hyon Cho,
and Srinivas Devadas

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Guaranteed in-order packet delivery using Exclusive Dynamic Virtual Channel Allocation

Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, and Srinivas Devadas
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{mieszko, ksshim, mhcho, devadas}@mit.edu

Abstract

In-order packet delivery, a critical abstraction for many higher-level protocols, can severely limit the performance potential in low-latency networks (common, for example, in network-on-chip designs with many cores). While basic variants of dimension-order routing guarantee in-order delivery, improving performance by adding multiple dynamically allocated virtual channels or using other routing schemes compromises this guarantee. Although this can be addressed by reordering out-of-order packets at the destination core, such schemes incur significant overheads, and, in the worst case, raise the specter of deadlock or require expensive retransmission.

We present Exclusive Dynamic VCA, an oblivious virtual channel allocation scheme which combines the performance advantages of dynamic virtual allocation with in-network, deadlock-free in-order delivery. At the same time, our scheme reduces head-of-line blocking, often significantly improving throughput compared to equivalent baseline (out-of-order) dimension-order routing when multiple virtual channels are used, and so may be desirable even when in-order delivery is not required. Implementation requires only minor, inexpensive changes to traditional oblivious dimension-order router architectures, more than offset by the removal of packet reorder buffers and logic.

1. Introduction

In-order packet delivery in a network is a widely assumed basis for a wide range of application protocols, e.g., file transfer and optimized cache coherence protocols [7, 10]. Basic dimension-order routing without virtual channels, an approach popular in network-on-chip (NoC) designs, always delivers packets between any two nodes in the order in which they were sent because all packets follow the same path and are stored in the same buffers. Because packets from different flows are buffered in the same queues, however, a single ill-behaved flow can overwhelm other flows and effectively block them even if they are destined for a different egress port, a phenomenon known as head-of-line blocking.

To address this, multiple virtual channels (VCs) can be used on each link, either allocated statically (so that each flow uses only a specific VC in each node) or dynamically (so that each packet may be assigned to any available VC). While static VC allocation can ensure that each flow uses a single VC per node and packets arrive in order, head-of-line blocking remains a problem: when the number of flows exceeds the number of VCs and multiple flows must share one VC, that VC can be overwhelmed by a single flow even though all other VCs carry no traffic. Moreover, efficient static VC allocation requires a priori knowledge of the application's traffic patterns [17], a reasonable assumption for fixed-application chips but an unrealistic requirement for general-purpose NoCs. Dynamic VC allocation can adapt VC usage to the traffic pattern by allocating available VCs to packets as they arrive, but, for certain traffic patterns, performance can still suffer significantly because of head-of-line blocking. Moreover, dynamic VC allocation forfeits the advantage of in-order packet delivery because two packets from the same flow may be assigned to different VCs in the same node and leave the VCs in an order different from their arrival sequence; effectively, dynamic VC allocation creates multiple virtual paths for each flow.

When the routing algorithm itself directs a single flow via multiple paths (either via different sequences of nodes or via dynamically allocated VCs in the same sequence of nodes), in-order delivery can be accomplished by resorting to packet reordering. Each packet is tagged with a sequential serial number, and any packets that arrive out of order are stored in a reorder buffer at the destination node until all of their predecessors have been received. To avoid deadlock scenarios in the event a reorder buffer fills up, the network must be able to dynamically limit the number of outstanding packets of a given flow, either by dropping and resending packets, or by sending packets only after enough preceding packets have been delivered; either case requires some form of an acknowledgement protocol.

While this approach works well when the processing element (PE) at each node is significantly faster than the network (as is the case, for example, with multicomputer networks like the Internet), it is less appropriate for NoCs where the on-chip network fabric is very fast compared to the PE. While the reordering protocol can be implemented in hardware to improve performance in such cases, the amount and complexity of the necessary logic can be daunting.

In this paper, we propose Exclusive Dynamic Virtual Channel Allocation (EDVCA), a VC allocation scheme which combines the benefits of static and dynamic VC allocation regimes by ensuring that a flow is traveling via at most one path *at any one instant*. When combined with multi-VC dimension-order routing, our method guarantees deadlock-free in-order packet delivery at a fraction of the hardware cost and complexity of packet reordering approaches and without the overhead of including a serial number in each packet. Moreover, EDVCA significantly improves network performance for traffic patterns susceptible to head-of-line blocking, while offering performance equivalent to standard dynamic VC allocation on other traffic. In the following sections, we outline our scheme, detail implementation differences relative to a baseline oblivious virtual-channel router design, and analyze performance via extensive cycle-accurate simulation with synthetic as well as realistic traffic patterns.

2. Related work

2.1. Routing algorithms

Perhaps the simplest routing scheme is dimension-ordered routing (DOR), which applies to a broad class of networks, including the 2D mesh we consider here [5]. Packets traverse each dimension in a predefined order, traveling along one dimension until they have reached the destination node coordinate along that dimension, at which point they switch to the next dimension. In the simplest, one-channel version, DOR delivers packet in order within each flow, although implementations with dynamically allocated VCs forfeit this guarantee because packets may pass each other in neighboring VCs. Although low implementation complexity and in-order packet delivery (under some virtual channel allocation schemes) have made DOR a very popular choice in the on-chip network design space, its simplicity comes at the cost of poor worst-case and average-case throughput for mesh networks.

By balancing traffic between the XY and YX variants of DOR routing, O1TURN [16] not only guarantees provable worst-case throughput but also matches average-case behavior of ROMM for both global and local traffic, and generally performs very well in practice with very little extra hardware cost and no transmission overhead. Since packets in the same flow may experience different congestion characteristics along the two possible routes, however, O1TURN may deliver packets out of order.

Valiant and Brebner proposed a routing scheme that chooses an intermediate node at random and employs DOR to route first to the intermediate node and then from there to the destination [21]. While this algorithm achieves optimal worst-case throughput by spreading all traffic throughout the network, it sacrifices average-case behavior and latency, since even traffic between adjacent nodes may incur significant delays in traveling to and from the intermediate node; in addition, since different packets from the same flow may use different intermediate nodes, they may be delivered out of order.

ROMM [13, 14] applies the Valiant algorithm to minimum routing, limiting intermediate node choices to the minimum rectangle defined by the source and destination nodes. Although ROMM guarantees minimum routes, its load balancing is not optimal, and it may saturate at a lower throughput than DOR in 2D torus networks [20] and 2D mesh networks [16]. While increasing the number of phases can reduce congestion, it comes at the cost of additional hardware complexity and additional virtual channels. Like Valiant, ROMM may deliver packets out of order.

Classic adaptive routing schemes include the turn routing methods [8] and odd-even routing [3]. These are general schemes that allow packets to take different paths through the network while ensuring deadlock freedom but do not specify the mechanism by which a particular path is selected. An adaptive routing policy determines what path a packet takes based on network congestion. Many policies have been proposed (e.g., [4, 11, 18, 19, 9]). Since adaptive routing algorithms alter the route in response to network conditions, they can also deliver packets out of order.

2.2. In-order packet delivery

Few routing scheme designs address out-of-order packet delivery. Within the Network-on-Chip (NoC) context, Murali et al [12] describe a multi-path in-order scheme where sequentially numbered packets belonging to a given flow are delayed at switches where distinct paths used by the same flow join (or cross). This scheme relies on a static assignment of flows to links; moreover, their re-ordering method contemplates only packets within one flow and either does not consider the possibility of deadlock when separate flows block each other or makes an unrealistic assumption of a private virtual channel for each flow.

More generally, ensuring in-order delivery via buffering out-of-order packets at the destination node and reordering them (and, if necessary, dropping and retransmitting) has been around since the dawn of computer networking, and is employed, for example, in the Transmission Control Program [2, 1], the precursor of the ubiquitous Transmission Control Protocol [15]. TCP combines destination-buffered reordering with window-based flow control and acknowledgements piggybacked on return packets.

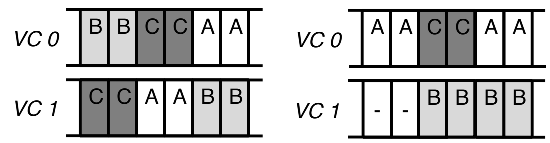
3. Exclusive Dynamic VC Allocation

In a nutshell, exclusive dynamic VCA prevents packets from any single flow from using more than one VC at a given ingress *at any given instant*. When a snapshot of the network is examined, the system *appears* as if VCs had been statically allocated: packets from any given flow use no more than one VC per ingress, and seem to travel via a single path, which ensures in-order packet delivery and reduces head-of-line blocking. At the same time, the VC being used for a specific flow at a given ingress can change over time, and when the network state is examined over a longer period, flows may appear to be using multiple VCs at each ingress port, spreading incoming traffic among available VCs.

In what follows, we assume a standard ingress-queued virtual-channel router with wormhole routing and credit-based inter-link flow-control [6]. In such designs, each packet arriving at an ingress port is immediately queued in a VC buffer, and forwarded via five steps: route computation (RC), virtual channel allocation (VCA), switch allocation (SA), and switch traversal (ST), sometimes implemented as separate pipeline stages for efficiency. All flits in a packet are forwarded contiguously, so the first two stages (RC and VCA) only perform computation for the head flit of each packet, returning cached results for the remaining flits.

Our scheme differs only in the VCA stage: when allocating a next-hop VC to a packet from flow f, we follow the rules below:

- if no next-hop VC contains packets from f, assign the packet to any available VC; if no VCs are available, stall the packet and try to allocate again in the next cycle (emulates dynamic VCA)
- if some next-hop VC v already contains packets from f, and v is available, assign the packet to v; if v is not available, stall the packet and try to allocate again in the next cycle (emulates static allocation of f to v)



(a) dynamic VCA (b) exclusive dynamic VCA

Figure 1. Comparison of VCA schemes

Figure 1 illustrates how our scheme might allocate virtual channels for packets from three hypothetical flows, *A*, *B*, and *C*, for a two-VC ingress port. Traditional dynamic allocation might assign packets from each flow to both VCs, as shown in Figure 1(a); exclusive dynamic VCA, on the other hand, will assign packets from one flow to only one VC, as shown in Figure 1(b). Thus, when the third packet in flow *B* arrives, it is assigned VC 1 because VC 1 already has packets from *B*; similarly, the third packet in flow *A* is assigned VC 0. When the third *C* packet arrives, it must wait either until VC 0 has space (in which case it can go into VC 0) or until all other *C* packets in VC 0 has been forwarded (in which case it can go into either VC). Note that, while Figure 1(b) only shows a snapshot at a particular instant and the VC assignment might change at a later time (for example, with VC 0 and VC 1 reversed), exclusive dynamic VCA will never result in the situation shown in Figure 1(a).

When EDVCA is implemented in a routing regime where a given flow always follows the same sequence of nodes (e.g., dimension-order routing), it guarantees that all packets within one flow will be forwarded and delivered in the original order. This is of course the case for a single VC, since, at each node, all packets are buffered in the same ingress queue and depart in arrival order; it is also true for static VCA because all packets *in the same flow* stay in the same queue. Dynamic VCA gives up in-order delivery, since it allows packets from the same flow to be buffered in different VCs, and packets in different VCs may pass each other; for example, in Figure 1(a), the packets from flow *A* could depart in any of six possible orders. We restore the in-order delivery guarantee in EDVCA by observing that the static-VCA condition above needs to be satisfied only at any given instant; that is, packets are still delivered in order if, within each node, all packets in the same flow are buffered in the same VC *at any given time*.

While our motivation in this paper is in-order packet delivery in NoCs and we focus on applying EDVCA to dimension-order routing on a mesh, the scheme is independent of network geometry and route selection, and can be directly applied to other oblivious routing algorithms (e.g., O1TURN [16], Valiant [21], or ROMM [13, 14], more sophisticated table-based static routing [17]), or adaptive routing schemes like turn methods [8] or odd-even routing [3].

Finally, our VC allocation scheme is also free of deadlock provided the underlying routing regime is also deadlock-free. The only new dependencies are between packets waiting to be forwarded and the next-hop VCs they are targeting, a kind of dependency already present in any kind of ingress-queued router.

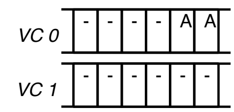
In addition to EDVCA, in which each flow exclusively uses a VC at any given time, we also considered an orthogonal scheme where each VC may only contain one flow at any given time (but gives up in-order guarantees because packets from the same flow may use more than one VC), and a scheme that combines both constraints (thus guaranteeing in-order delivery). While both schemes still outperformed DOR in most of our experiments, they did not perform quite as well as EDVCA; since flows tended to monopolize VCs and cause unfairness, we judged these variants to be impractical and do not detail them here.

4. Implementation cost

Ensuring in-order packet delivery in a fast on-chip network incurs some additional cost in all but the most basic routing schemes (e.g., single-VC DOR); for example, a store-and-reorder scheme with good performance would require both significant buffer space at the destination and, in order to ensure that the destination buffers do not overflow, either end-to-end flow control or retransmission capability, both of which require additional memory space at the source. In comparison, our scheme requires only a modicum of additional hardware, which we detail by highlighting the differences from a classical virtual channel router [6] with dynamic VC allocation.

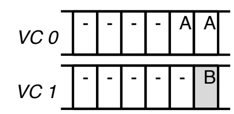
The additional hardware is required to ensure that packets from a single flow will not be buffered in two VCs at the same time; that is, during the VCA stage, the router must check whether any VCs at the next-hop node already contain the relevant flow, and, if so, restrict itself to that VC. This can be accomplished with a per-flow table, where the entry for each flow lists the currently assigned VC (if any), and the number of flits from that flow remaining in that VC; Figure 2 illustrates how

Flow ID	VC	# of flits
A	**0**	**2**
B	–	–
C	–	–



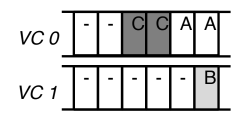
(a) Two flits in flow *A* departs for next-hop VC 0

Flow ID	VC	# of flits
A	0	2
B	**1**	**1**
C	–	–



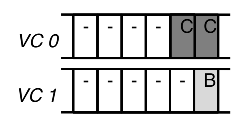
(b) One flit in flow *B* departs for next-hop VC 1

Flow ID	VC	# of flits
A	0	2
B	1	1
C	**0**	**2**



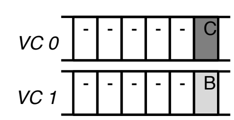
(c) Two flits in flow *C* depart for next-hop VC 0

Flow ID	VC	# of flits
A	**–**	**–**
B	1	1
C	0	2



(d) Two flits in flow *A* are forwarded from next-hop VC 0

Flow ID	VC	# of flits
A	–	–
B	1	1
C	**0**	**1**



(e) One flit in flow *C* is forwarded from next-hop VC 0

Figure 2. Tracking remote VC contents

such a table might be updated when packets depart from the current node for the next hop, and when they are forwarded on from the next-hop node. For each head flit, the VCA stage locates the next available next-hop VC (same as in traditional dynamic VCA), and, in parallel, queries the table. If the table lookup returns no entry for the flow, the next available VC is used; otherwise, the VC from the table lookup is used if it is available, or no assignment is made if the VC is busy. Assuming that the latency of the VCA stage is comparable to that of the table query, the only additional latency comes from a small multiplexer arbitrating between the traditional VCA result and the table lookup.

While at first blush it might appear that in a system with many flows such a table might have to be quite large, observe that only flows with any flits in the next-hop VCs need to be stored. This is relatively few flows:

for a node with 8 eight-flit queues per ingress, at most 64 different flows can be buffered at each ingress.[1] Furthermore, the table can be smaller than the number of simultaneously buffered flows: if the table becomes full, the next packet for an unknown flow stalls until there is space in the table and performance degrades gracefully. In any event, a table of this size can be efficiently implemented using a content-addressable memory (CAM) addressed by flow ID, one for each ingress if each flow has only one entry port (e.g., in DOR), or one bigger table for each node if flows can arrive from any direction (e.g., in some adaptive routing schemes).

The only remaining overhead stems from tracking the number of flits for each flow in the next-hop VCs (Figure 2). This can be handled by modifying the existing credit update system, which allows a node to keep track of the number of free slots in each next-hop VC. In the original (non-EDVCA) version, the router decrements a per-remote-VC credit count whenever a flit departs for a next-hop VC and increments it when the neighboring node removes a flit from that VC and sends a credit update message. Since EDVCA tracks remote VC contents for each flow (Figure 2), the credit update messages carry a flow ID instead of a VC ID, and the corresponding VC IDs are locally retrieved from the table and used to update the relevant per-VC credits; thus, the send events in Figures 2(a)–2(c) cause credit decrements for the relevant VCs, and the update events in Figures 2(d)–2(e) trigger credit increments. While sending flow IDs instead of VC IDs in every credit update message does increase the number of bits and therefore the bandwidth used for credit updates, the few extra wires are cheap in terms of area and do not affect the size of the crossbar switch; furthermore, if desired, the wire count can be decreased significantly by sending credit updates less often, with corresponding graceful decrease in performance.

These small overheads compare favorably with the resources and logic required to implement a typical store-and-reorder scheme for in-order delivery. Unlike reorder buffer size, the additional table memory does not grow with maximum packet size, and the additional VC allocation and credit update logic is much simpler than the logic needed to reorder, acknowledge, and possibly retransmit packets.

5. Results

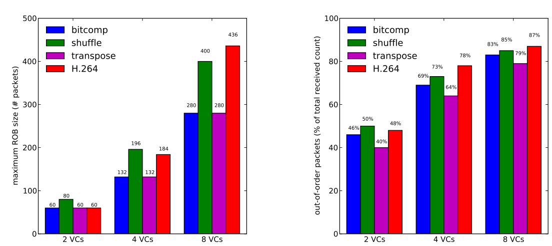
We have evaluated the performance of exclusive dynamic VCA via extensive simulation on synthetic benchmarks as well as a load profile obtained from a parallel implementation of an H.264 video decoder, and report the results below.

Topology	8×8 2D mesh
Routing	DOR–XY, O1TURN
Link bandwith	2, 4, 8 flits/cycle
VC alloc	Dynamic, EDVCA
VC output mux	None, 1, 2, 4
Per-hop latency	1 cycle
VCs per port	1, 2, 4, 8
VC buffer size	8 flits
Average packet length	2, 8, 32 flits
Traffic workload	transpose, shuffle, bit-complement, H.264 decoder profile
Burstiness model	Markov modulated
Warmup cycles	10,000
Analyzed cycles	100,000

Table 1. Network configuration summary

5.1. Experimental setup

For our experiments, we used an in-house cycle-accurate NoC simulator. The simulator implements a standard ingress-queued virtual-channel router [6]. To mitigate crossbar cost with routing schemes that require multiple VCs to avoid deadlock, a VC output multiplexer can optionally choose a subset of the VCs at each ingress, and present the chosen subset for switch allocation. To avoid unfairness effects resulting from a particular combination of round-robin strategy and packet arrival rate, VCs in switch and VC allocation are considered in random order and greedily matched. While the simulator can be configured for any desired geometry and a host of oblivious routing and virtual channel allocation schemes, we focused our experiments on

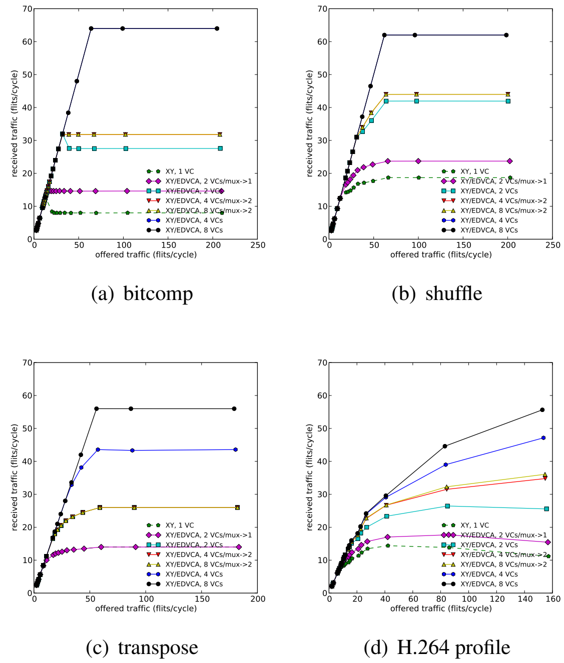


(a) Maximum number of packets from any single flow in any reorder buffer

(b) Percentage of packets received out of order for that flow

Figure 3. Reorder costs, XY + dynamic VCA

[1]or half of that, assuming a minimum packet length of one routing (head) flit and one data flit



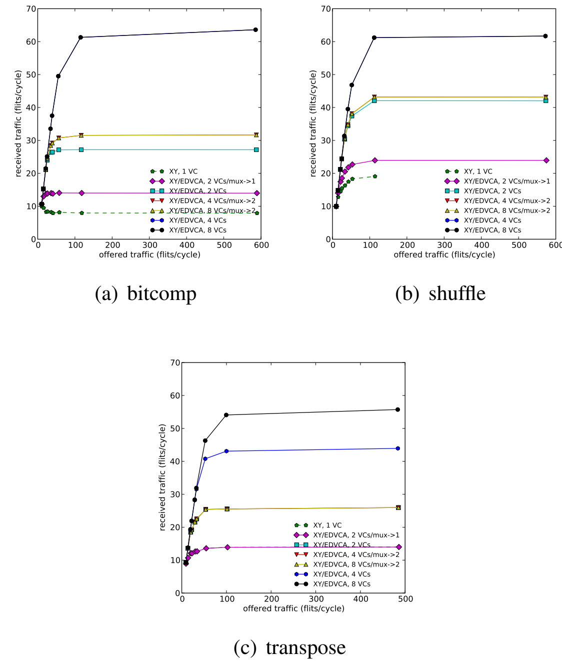
(a) bitcomp

(b) shuffle

(c) transpose

(d) H.264 profile

Figure 4. In-order throughput



(a) bitcomp

(b) shuffle

(c) transpose

Figure 5. In-order burst throughput

an 8×8 mesh network under XY and O1TURN routing. To estimate the impact of out-of-order packets, we implemented a store-and-reorder strategy, although reorder buffer sizes are not limited and so retransmission is never necessary.

Table 1 summarizes the configurations used for our experiments. Although we analyzed performance for a variety of link bandwidths and flit sizes, for clarity of exposition we avoid plotting data that are similar, and show only results for link bandwidth of 8 flits/cycle and packet sizes of 2 flits (for the synthetic benchmarks with 1, 2, and 4 VCs) and 8 (for 8 VCs and the H.264 profile load). For the synthetic benchmarks, we simulated constant and Markov-chain-modulated bursty traffic.

5.2. Out-of-order packet delivery

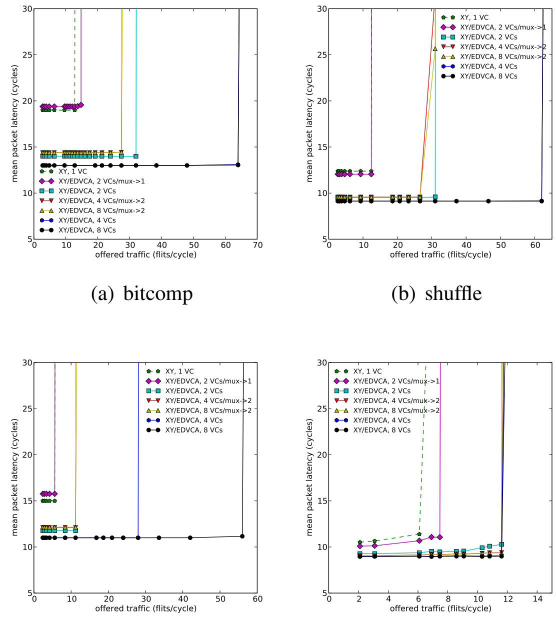
To understand the cost of a store-and-reorder in-order implementation in DOR[2] with dynamic VC allocation, we asked how many packets arrived out of order (that is, before all of their predecessors were delivered).

Figure 3(a) shows, for each load and VC count, the highest number of packets from any one flow simultaneously waiting in any reorder buffer in any of our experiments; that is, it indicates the minimum reorder buffer size required to avoid dropping and retransmitting packets in all experiments. Figure 3(b) shows the percentage of packets that arrived out of order for the corresponding flow in the corresponding experiment. The buffer size observations confirm that out-of-order packet arrival is a significant problem across a wide variety of loads, and shows that per-flow reorder buffers at each destination must be relatively large to avoid expensive retransmission. The high percentage of packets received out of order indicates that the reorder buffer and reordering logic must operate at close to line rate, effectively excluding any software-level solution. Since one such buffer may have to be implemented for each flow arriving at a given destination, and the efficiency demands would require very fast storage, the cost of reorder buffer space alone in a store-and-reorder scheme would be significant.

5.3. Comparison among in-order variants

To examine the performance potential when in-order delivery is a must, we compared EDVCA with different number of VCs against the only other network-level routing that conserves packet order, DOR with one VC (Figures 4–5). We found that EDVCA performance scaled with the number of VCs, and so allows in-order delivery to be implemented at any chosen cost vs. per-

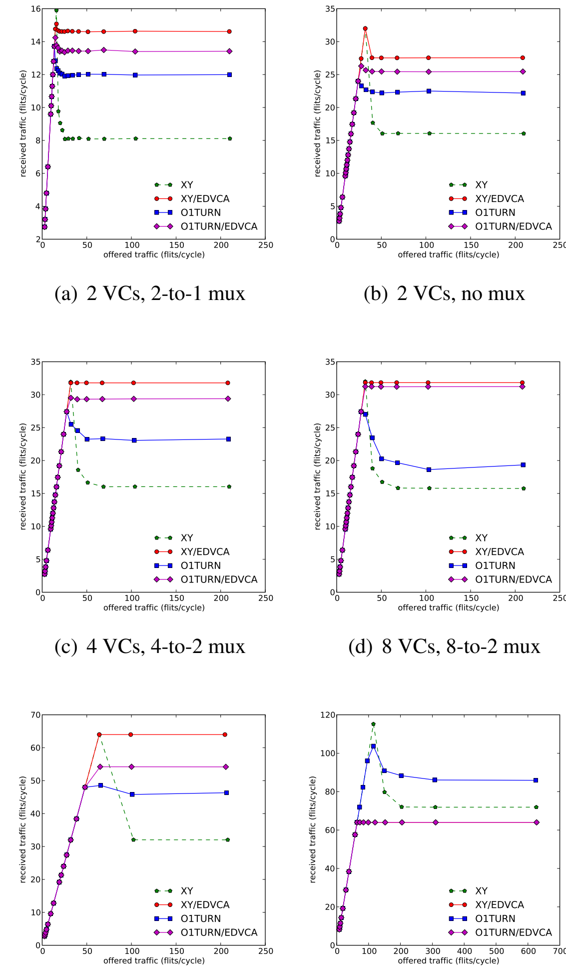
[2] We considered only DOR when examining out-of-order packet rates to ensure a fair comparison with EDVCA, since EDVCA cannot guarantee in-order delivery under O1TURN routing.



(a) bitcomp (b) shuffle

(c) transpose (d) H.264 profile

Figure 6. In-order latency



(a) 2 VCs, 2-to-1 mux (b) 2 VCs, no mux

(c) 4 VCs, 4-to-2 mux (d) 8 VCs, 8-to-2 mux

(e) 4 VCs, no mux (f) 8 VCs, no mux

Figure 7. Same HW: bit-complement

formance tradeoff. Although chip area and therefore system cost increases with the number of VCs, a large portion of this is due to increase in the size of the crossbar switch, which can be mitigated by placing a multiplexer between the VCs and the crossbar; for example, two-VC EDVCA with such a multiplexer (violet diamonds in Figure 4) is much closer in cost to one-VC DOR, but still offers better performance. The higher throughput enabled by using more virtual channels with EDVCA also tends to lower latency (Figure 6).[3].

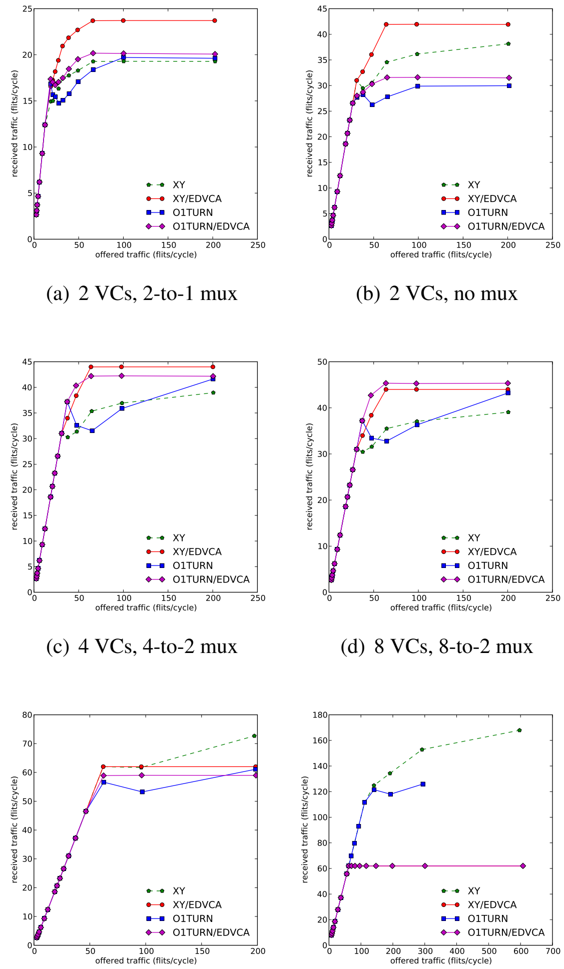
5.4. Comparison on equivalent hardware

We next examined performance when the system cost was held constant, and in-order delivery was not considered. XY/EDVCA still tended to outperform both dynamic-VCA XY and O1TURN[4] under heavy load in most scenarios where load caused VC contention (Figures 7 and 8), except for the highly asymmetric transpose pattern (Figure 9). When the number of VCs sufficiently exceeds flow load, EDVCA suffers from the restriction that each flow must only travel via one VC, and is outperformed by XY and O1TURN, which can use all VCs; the H.264 profile throughput (Figure 10) offers a good example. Even here, however, when VCs are multiplexed prior to competing for the crossbar, as is the case in many routers, EDVCA performs better than XY routing (e.g., Figure 10(d)).

Although in the regimes where EDVCA offers higher throughput when the network is oversubscribed it also improves latency for *delivered* packets when the network is oversubscribed, this is irrelevant to an end-to-end packet latency measurement as in Figure 6, which would be dominated by the waiting time of packets that are never sent because the network cannot handle any more traffic. Instead, end-to-end latency is only relevant in a small regime of load factors, at the point

[3] Our latency measures encompass the end-to-end delay between, the time the packet is queued in the input buffer and the time the last flit is received at the destination, and include the time spent waiting to be injected into the network at the source core.

[4] Note that O1TURN is at a disadvantage under an equivalent-hardware comparison, as it requires twice as many VCs as DOR to avoid deadlock.



(a) 2 VCs, 2-to-1 mux

(b) 2 VCs, no mux

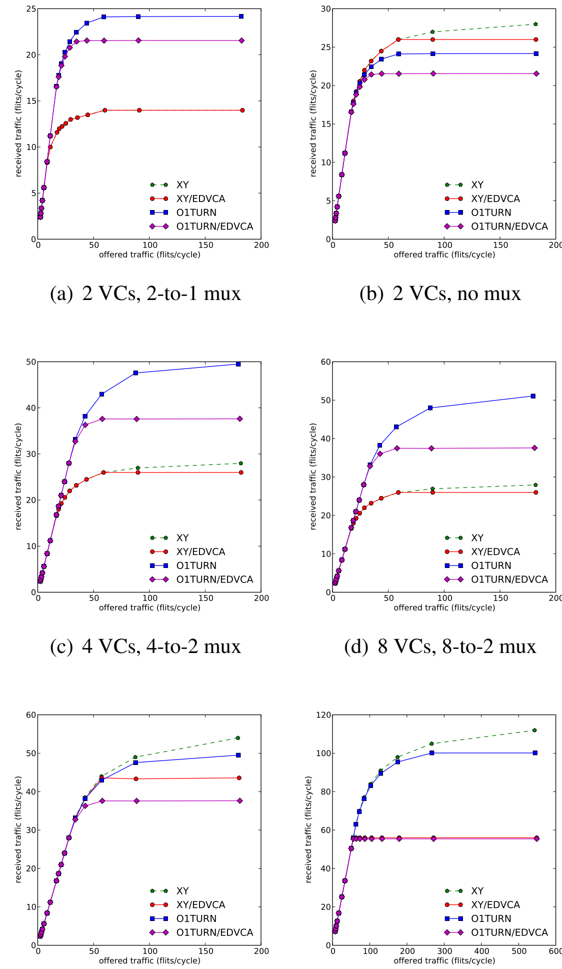
(c) 4 VCs, 4-to-2 mux

(d) 8 VCs, 8-to-2 mux

(e) 4 VCs, no mux

(f) 8 VCs, no mux

Figure 8. Same HW: shuffle



(a) 2 VCs, 2-to-1 mux

(b) 2 VCs, no mux

(c) 4 VCs, 4-to-2 mux

(d) 8 VCs, 8-to-2 mux

(e) 4 VCs, no mux

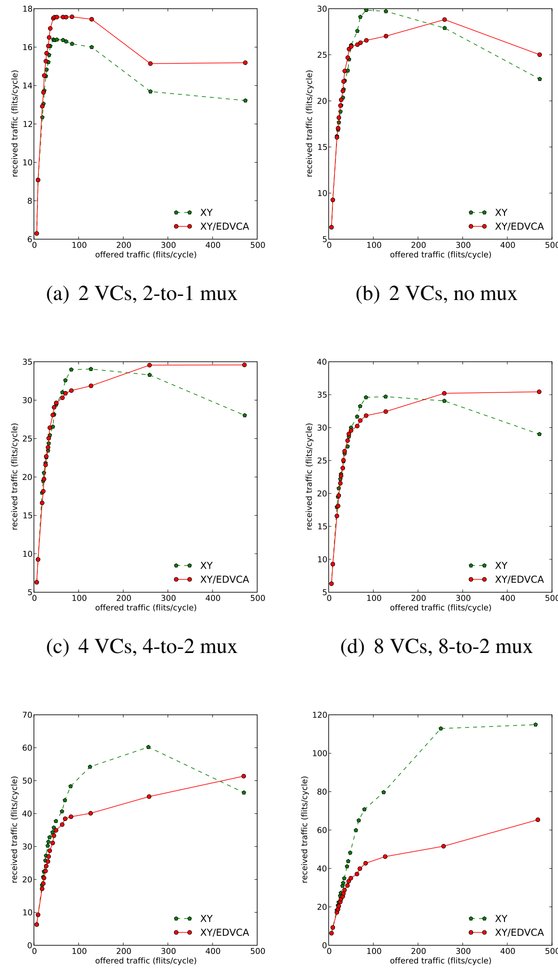
(f) 8 VCs, no mux

Figure 9. Same HW: transpose

where the network becomes overloaded and stops delivering traffic at the rate of 100%; in the throughput plots in Figures 7–10, this is the point at which the initial offered-vs-received traffic line decreases in slope. Figure 11 illustrates this point: in Figure 11(a), EDVCA matches the underlying latency of DOR because 100% packet delivery begins to fail at the same injection rate, while in Figure 11(b) DOR offers low latency longer because it fails at a higher injection rate (cf. Figure 7).

6. Conclusion

Although applications that require packets to arrive in the order in which they were sent are ubiquitous, guaranteeing in-order packet delivery has received comparatively little attention in routing algorithm design, and, with the exception of single-VC dimension-order routing and static VC assignment, has generally been relegated to a higher level of abstraction. As ultra-fast on-chip networks become common, however, buffer-based packet reordering can become a significant bottleneck. By combining the benefits of dynamic and static virtual channel allocation schemes, Exclusive Dynamic Virtual Channel Allocation allows dynamic VC assignment while guaranteeing in-order packet delivery at the network transport level. Ensuring in-order delivery at the network level obviates the need for expensive buffers and retransmission logic, promising better performance at a lower cost than a traditional higher-level store-and-reorder scheme in the niche of fast on-chip networks.

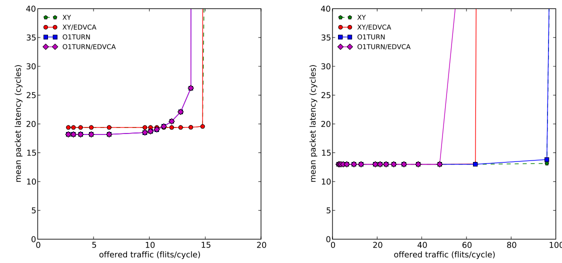


(a) 2 VCs, 2-to-1 mux (b) 2 VCs, no mux

(c) 4 VCs, 4-to-2 mux (d) 8 VCs, 8-to-2 mux

(e) 4 VCs, no mux (f) 8 VCs, no mux

Figure 10. Same HW: H.264 profile



(a) 2 VCs, 2-to-1 mux (b) 8 VCs, no mux

Figure 11. Same HW latency: bitcomp

References

[1] V. G. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, Internet Engineering Task Force, December 1974.

[2] V. G. Cerf and R. E. Kahn. A Protocol for Packet Network Communication. *IEEE Trans. Comm.*, 22:637–648, May 1974.

[3] Ge-Ming Chiu. The Odd-Even Turn Model for Adaptive Routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[4] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 04(4):466–475, 1993.

[5] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Computers*, 36(5):547–553, 1987.

[6] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[7] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[8] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, 1994.

[9] P. Gratz, B. Grot, and S. W. Keckler. Regional Congestion Awareness for Load Balance in Networks-on-Chip. In *In Proc. of the 14th Int. Symp. on High-Performance Computer Architecture (HPCA)*, pages 203–214, February 2008.

[10] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann, September 2006.

[11] H. J. Kim, D. Park, T. Theocharides, C. Das, and V. Narayanan. A Low Latency Router Supporting Adaptivity for On-Chip Interconnects. In *Proceedings of Design Automation Conference*, pages 559–564, June 2005.

[12] S. Murali, D. Atienza, L. Benini, and G. De Micheli. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of DAC 2006*, pages 845–848, July 2006.

[13] Ted Nesson and S. Lennart Johnsson. ROMM Routing: A Class of Efficient Minimal Routing Algorithms. In *in Proc. Parallel Computer Routing and Communication Workshop*, pages 185–199, 1994.

[14] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'95*, pages 275–287, 1995.

[15] J. Postel. Transmission Control Protocol. RFC 793, Internet Engineering Task Force, September 1981.

[16] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-Optimal Worst-Case

Throughput Routing for Two-Dimensional Mesh Networks. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA 2005)*, pages 432–443, 2005.

[17] K. S. Shim, M. H. Cho, M. Kinsy, T. Wen, M. Lis, G. E. Suh, and S. Devadas. Static Virtual Channel Allocation in Oblivious Routing. In *Proceedings of the* 3^{rd} *ACM/IEEE International Symposium on Networks-on-Chip*, May 2009.

[18] Arjun Singh, William J. Dally, Amit K. Gupta, and Brian Towles. GOAL: a load-balanced adaptive routing algorithm for torus networks. *SIGARCH Comput. Archit. News*, 31(2):194–205, 2003.

[19] Arjun Singh, William J. Dally, Brian Towles, and Amit K. Gupta. Globally Adaptive Load-Balanced Routing on Tori. *IEEE Comput. Archit. Lett.*, 3(1), 2004.

[20] Brian Towles and William J. Dally. Worst-case traffic for oblivious routing functions. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–8, 2002.

[21] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, 1981.