Allocator Implementations for Network-on-Chip Routers

Daniel U. Becker
Computer Systems Laboratory
Stanford University
dub@cva.stanford.edu

William J. Dally
Computer Systems Laboratory
Stanford University
dally@cva.stanford.edu

ABSTRACT

The present contribution explores the design space for virtual channel (VC) and switch allocators in network-on-chip (NoC) routers. Based on detailed RTL-level implementations, we evaluate representative allocator architectures in terms of matching quality, delay, area and power and investigate the sensitivity of these properties to key network parameters. We introduce a scheme for sparse VC allocation that limits transitions between groups of VCs based on the function they perform, and reduces the VC allocator's delay, area and power by up to 41%, 90% and 83%, respectively. Furthermore, we propose a pessimistic mechanism for speculative switch allocation that reduces switch allocator delay by up to 23% compared to a conventional implementation without increasing the router's zero-load latency. Finally, we quantify the effects of the various design choices discussed in the paper on overall network performance by presenting simulation results for two exemplary 64-node NoC topologies.

1. INTRODUCTION

Modular, communication-centric design methodologies employing networks-on-chip (NoCs) will become crucial to keeping design complexity for future chip multiprocessors (CMPs) and systems-on-chip (SoCs) manageable in the face of continuously increasing transistor budgets [1, 3]. As the number of architectural blocks that are integrated on a single chip continues to rise, overall system performance and cost become increasingly dependent on the efficiency of the NoC implementation.

In addition to network-level design choices like topology, routing function and flow control, router microarchitecture represents a major factor in determining network performance. One important design parameter for NoC routers is the choice of switch allocator and—in networks employing virtual channels (VCs) [2]—VC allocator. The quality of matchings generated by the latter can affect the average time head flits have to wait before being assigned an output VC, preventing forward progress for subsequent flits of the same packet and thereby increasing average buffer utilization. Furthermore, when speculative switch allocation [17] is employed, the quality of VC allocation affects the number of misspeculations that are caused by flits speculatively gaining access to the crossbar but failing to acquire a VC. The switch allocator, on the other hand, schedules flits that are buffered at the router's inputs for traversal through the crossbar; consequently, router performance under load is strongly dependent on the quality of the matchings it generates between requests and available crossbar time slots.

Individual allocator implementations represent different tradeoffs between matching quality and delay. Compared to traditional long-haul and system-level networks, performance in NoCs is typically much more sensitive to packet latency; this mandates the use of relatively shallow router pipelines and single-cycle allocation schemes. At the same time, NoCs are usually subject to tight cycle time, area and power constraints. Consequently, designers must select allocator implementations that maximize matching quality subject to these constraints.

Allocation in VC-based NoCs has been addressed in a number of prior research contributions:

Peh and Dally [17] present analytical delay models for VC and switch allocators and propose a speculative switch allocation mechanism. The delay models are derived based on gate-level schematics; however, the authors only consider separable input-first allocators in their analysis.

Mullins et al. [15] propose a technique for reducing the delay of separable allocators by pre-computing arbitration decisions one cycle in advance, and suggest a scheme for reducing VC allocation delay based on a free VC queue. Park et al. [16] introduce a mechanism that prioritizes flits traveling along frequently used paths; furthermore, they suggest a method to allow such flits to bypass the switch allocation pipeline stage entirely by propagating arbitration requests ahead of the actual flit. Kumar et al. [12] describe a switch allocation scheme that dynamically transitions between input- and output-first operation based on network load, and prioritizes subsequent flits from the same packet in an effort to minimize the number of routers at which a given packet occupies resources. Kim et al. [11] introduce an efficient switch allocation scheme for their proposed row/column decoupled router architecture; however, it is not directly applicable to different router designs. In all four papers, the authors focus their analysis on a specific network configuration, and do not evaluate how performance and cost of the proposed mechanisms scale with the network radix and the number of VCs.

Mukherjee et al [14] compare the performance of three different switch allocator implementations in the context of a chip-to-chip processor interconnection network. The study assumes a deeply pipelined router, and two of the allocators considered require multiple cycles to generate a schedule; both factors would be undesirable in a latency-critical NoC scenario.

In the present work, we evaluate and compare RTL-level implementations of three representative switch and VC allocator architectures in terms of matching quality, critical path delay, area and power. We investigate the sensitivity of these properties to key network parameters, and suggest two microarchitectural improvements: Sparse VC allocation significantly reduces the VC allocator's logic complexity by limiting transitions between groups of VCs; furthermore, the critical path delay of a speculative switch allocator can be improved by choosing a pessimistic implementation that sacrifices speculation efficiency under load, but does not increase the router's zero-load latency. In order to quantify how allocator design choices affect overall network-level performance, we present simulation results for two exemplary 64-node NoC topologies across a set of different network parameters and load conditions. We find that the sensitivity of network performance to the quality of switch allocation grows with the router radix and the number of VCs, and that overall performance is largely insensitive to the choice of VC allocator.

The remainder of the paper is organized as follows: Section 2 provides a brief summary of basic allocation concepts. Section 3 outlines our evaluation methodology. Section 4 describes VC allocator implementations and introduces a scheme for sparse VC allocation. Section 5 discusses switch allocator implementations and improvements to the speculation mechanism. Finally, Section 6 concludes the paper.

2. BACKGROUND

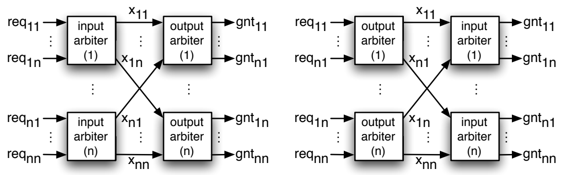
This section gives a brief summary of the basic principles of allocation; a more thorough introduction is available in [4].

An allocator performs a matching between resources and requesters; i.e., an allocator assigns the former to the latter subject to three basic constraints: Resources are only granted to requesters if a corresponding request exists, at most one resource is assigned to each requester, and each resource is assigned to at most one requester. Any assignment that satisfies these constraints is called a *matching*. Matchings in which no further resource assignment can be made without first undoing an existing one are called *maximal*, and those with the highest number of assignments possible are called *maximum* matchings.

Clearly, in order to maximize resource utilization, it is desirable for an allocator to produce matchings that are close to maximum. In practice, however, there is a tradeoff between matching quality on the one hand, and delay, area and power constraints that limit the allocator's logic complexity on the other hand.

2.1 Separable Allocation

A separable allocator generates a valid matching by decomposing allocation into arbitration across requesters and arbitration across resources. For *separable input-first allocation (sep_if)*, as shown in Figure 1(a), arbitration is first



(a) Input-first (sep_if). (b) Output-first (sep_of).

Figure 1: Separable allocators.

performed independently at each requester, and a single resource is selected for which to issue a request. A second round of arbitration is then performed at each resource to select a winning requester among all incoming requests. For *separable output-first allocation (sep_of)*, shown in Figure 1(b), requesters eagerly forward all of their requests to the associated resources, which again perform arbitration between all incoming requests. However, as multiple resources may select the same requester in the first stage, a second arbitration step is required, in which each requester chooses a winner among all resources that were assigned to it in the first stage.

In order to ensure fairness and to avoid traffic pattern dependent starvation, the input priorities for any given arbiter in either scheme's first arbitration stage are only updated if the grant it produces is also successful in the second arbitration stage and vice versa [13].

Arbiters—and thus separable allocators—can be designed such that their delay scales approximately logarithmically with the number of inputs, enabling relatively fast allocation even for high-radix routers. However, because arbitration across requesters and resources is performed independently, separable allocators are not guaranteed to produce maximal matchings. While multiple iterations can be performed to improve matching quality, tight delay constraints typically render this undesirable in the context of NoCs.

2.2 Wavefront Allocation

A *wavefront allocator (wf)* [19] works based on the principle that a conflict between two requests can only occur if the requester or the resource are identical for both, and that they can otherwise be granted independently. Consequently, if we represent the set of requests as a matrix, with rows corresponding to requesters, columns corresponding to resources, and non-zero matrix entries corresponding to requests, the wavefront allocator generates a matching by starting at an initial diagonal and granting all requests that fall under it. For each request that is granted, further requests in the same row or column are discarded, and allocation proceeds in the same fashion with the next diagonal, wrapping around at the end of each row and column as appropriate. This process is repeated until all diagonals have been serviced.

Weak fairness can be guaranteed by starting at a different initial diagonal every time allocation is performed; however, no fairness guarantees are provided beyond ensuring that all requests are eventually served.

Unlike separable allocators, wavefront allocators effectively consider requesters and resources simultaneously, and as a result are guaranteed to find maximal—although not necessarily maximum—matchings.

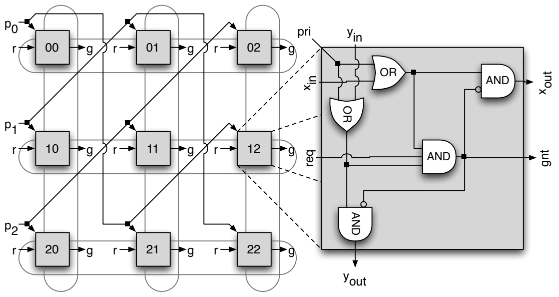


Figure 2: Wavefront allocator (wf).

Figure 2 shows how a wavefront allocator can be implemented as a regular array of simple base tiles, facilitating efficient full-custom implementations [5] with area that scales quadratically with the number of inputs and approximately linear delay. For synthesis-based implementations, however, the combinational loops formed by the x and y paths have to be broken in order to facilitate automated timing analysis and gate sizing. Noting that these paths are effectively disconnected at the currently active priority diagonal by the two OR gates, we build a loop-free implementation by replicating the array for each possible priority diagonal and selecting the grant matrix generated by the currently active one. Compared to a full-custom layout, this introduces a significant area overhead, and the required fanout and output multiplexers introduce additional logarithmic delay terms. A more area-efficient implementation that avoids combinational loops is presented in [9]; however, based on our experiments, the implementation described earlier tends to yield lower delay for the allocator sizes considered in this paper.

2.3 Maximum-Size Allocation

Conceptually, a maximum matching for a given set of requests and resources is readily found by performing successive iterations of an augmenting path algorithm [6]. However, while hardware implementations have been proposed that can perform one augmentation step per cycle [8], the associated complexity as well as the inherently iterative nature of generating a complete matching this way limit their applicability to NoC routers. Furthermore, maximum-size allocation inherently does not provide any fairness guarantees, and can cause starvation for individual requesters in the interest of maximizing overall throughput. Nevertheless, it provides a useful upper bound on matching quality that other allocators can be benchmarked against.

3. METHODOLOGY

Our evaluation of allocator architectures is based both on detailed RTL-level comparisons of individual allocators and on network-level simulation results.

We consider two exemplary 64-node network topologies and their corresponding allocator design points, identified by the number of ports *(P)* and the number of VCs *(V)*: An 8×8 mesh network *(mesh)* with a single network terminal per router represents a commonly-used low-radix ($P = 5$) NoC topology. As an example of a higher-radix topology, we furthermore consider a two-dimensional 4×4 flattened butterfly [10] network *(fbfly)* with a concentration factor of four ($P = 10$).

3.1 RTL-Based Evaluation

In order to be able to perform detailed cost evaluations and to gain insights about basic trends and tradeoffs that are independent of higher-level implementation details, we perform low-level comparisons of different allocator implementations based on isolated RTL models.

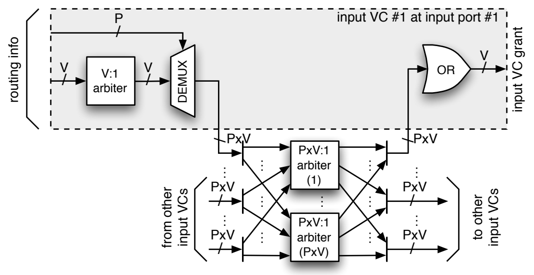
To this end, we have developed parameterized RTL implementations for the three allocator architectures considered. For each design point, we use Synopsys Design Compiler to find the minimum cycle time, the required cell area and the average power consumption when applying a default activity factor of 0.5 to all inputs; synthesis is performed using a commercial 45nm low-power standard cell library under worst-case process, voltage and temperature conditions (0.9V, 125°C).

Furthermore, we assess the quality of allocation provided by each allocator by performing open-loop simulations of the individual RTL implementations. We apply input sets of 10000 pseudo-random request matrices and count the resulting grants. A normalized metric—henceforth referred to as *matching quality*—is then obtained by dividing the total number of grants generated by each allocator implementation by the number of grants a maximum-size allocator would generate for the same sequence of requests.

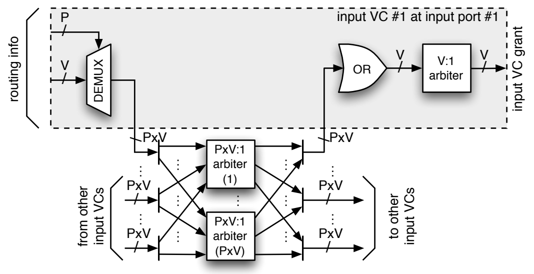
3.2 Network-Level Performance

We evaluate the impact of allocation on overall network performance by using a cycle-accurate network simulator to measure the average packet latency as a function of the flit injection rate. Network links have a latency of one cycle for the mesh, and one to three cycles for the flattened butterfly. We assume credit-based flow control and model an input-queued VC router design with a simple two-stage pipeline: VC and switch allocation take place in the first pipeline stage, and switch traversal occurs in the second stage. Input buffers are statically partitioned, with eight entries assigned to each VC. In order to reduce zero-load latency, we employ speculative switch allocation as introduced in [17]. Furthermore, lookahead routing [7] is used to pre-compute the routing decision for the next downstream router in parallel with the current router's VC allocation stage, effectively removing the routing logic from the critical path. While this approach reduces the depth of the router's pipeline and thus its zero-load latency, it does not favor the use of sophisticated adaptive routing algorithms, as any information required to make routing decisions must be available at the upstream router. Consequently, we implement dimension-order routing on the mesh network and the UGAL routing algorithm from [18] on the flattened butterfly.

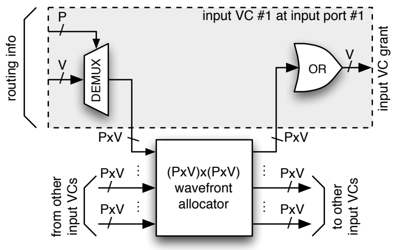
We present simulation results for uniform random traffic; additional simulation runs with other synthetic traffic patterns suggest that our conclusions are largely invariant to traffic pattern selection. Traffic comprises request and reply packets for read and write transactions. Read requests and write replies consist of a single flit, while read replies and write requests comprise a head flit and four flits containing payload data. Network terminals inject request packets into the network according to a geometric random process with configurable arrival rate. When such requests reach their destination terminal, a corresponding reply packet is generated in the next cycle and sent back to the source terminal; this takes priority over the injection of new request packets.



(a) Separable input-first allocator (sep_if).



(b) Separable output-first allocator (sep_of).



(c) Wavefront allocator (wf).

Figure 3: VC allocator block diagrams.

4. VC ALLOCATORS

In the following section, we evaluate three representative VC allocator implementations based on the architectures discussed in Section 2 and introduce a sparse VC allocation scheme that can significantly reduce logic complexity for common allocator configurations.

4.1 Implementation Details

Head flits buffered at any of the router's input VCs must be assigned a suitable output VC at the output port selected by the routing function before they can proceed through the router pipeline. Thus, the VC allocator performs a matching between $P\times V$ requesters and $P\times V$ resources, subject to the constraint that any output VCs requested by a given input VC at any given time share the same output port.

In the separable input-first implementation, shown in Figure 3(a), each input VC first determines which output VC at the destination output port to bid on. Requests are forwarded to a stage of $P\times V$-input arbiters at the output VCs as in the canonical implementation; these large $P\times V$-input arbiters can be implemented as tree arbiters—i.e., a stage of P V-input arbiters in parallel with a single P-input arbiter that selects among them—to reduce delay. Finally, grants for each input VC are grouped and reduced to a V-wide vector that indicates the granted output VC.

The separable output-first implementation is depicted in Figure 3(b). Here, each input VC forwards requests to all candidate output VCs at the destination port, where arbitration is again performed between all incoming requests. As a given input VC's requests may win arbitration at multiple output VCs, an additional stage of arbitration is needed after grouping and reducing grants to select a single winning VC.

Finally, the wavefront-based implementation, shown in Figure 3(c), consists of a canonical $P\times V$-input wavefront allocator, with additional logic for generating the $P\times V$-wide request vector for each input VC as in the separable output-first case, and reducing the $P\times V$-wide grant vectors to a V-wide vector as in the input-first case.

4.2 Sparse VC Allocation

VCs can serve a number of orthogonal purposes: Different types of packets, e.g. requests and replies, can be assigned to disjoint sets of VCs to prevent protocol-level deadlock at the network boundary; this partitions the total set of VCs into subsets corresponding to different *message classes*. In order to prevent deadlock scenarios caused by cyclic resource dependencies within the network, these subsets can be further partitioned into different *resource classes*, with transitions between the latter being restricted such as to enforce a partial order of resource acquisition. Examples of this approach include dateline routing in torus networks and two-phase routing as implemented in Valiant's algorithm [20] or the UGAL algorithm introduced in [18]. Finally, multiple VCs can be assigned to each subset; this improves network performance under load by eliminating head-of-line blocking, and allows for better channel utilization by sharing a physical link among multiple logical connections. Thus, we can express the total number of VCs as

$$V = M \times R \times C,$$

where M is the number of message classes, R is the number of resource classes, and C is the number of VCs assigned to each class.

In previous work, allocators have usually treated VCs in a largely uniform manner: The allocator logic is designed such that it can handle requests from any given input VC to the whole range of output VCs, and a bit vector supplied by the routing logic is used to constrain that range to a subset of allowable VCs at runtime. However, based on the following observations, we can exploit the assignment of VCs to different message and resource classes to statically constrain the set of candidate output VCs that a given input VC can generate requests for, and thus greatly reduce the VC allocator's logic complexity.

As packets are assigned to message classes according to the type of the packet itself, a packet's message class remains unchanged throughout its lifetime in the network. Thus, as packets never transition from one message class to another, we can partition the VC allocator into a set of smaller, completely separate VC allocators, each of which only needs to handle VCs corresponding to a single message class. For the separable allocator implementations, this enables us to reduce the number of ports for the input- and output-stage arbiters, as well as the number of output-side arbiters that each individual input VC connects to, by a factor of M. For the wavefront-based implementation, on the other hand, the

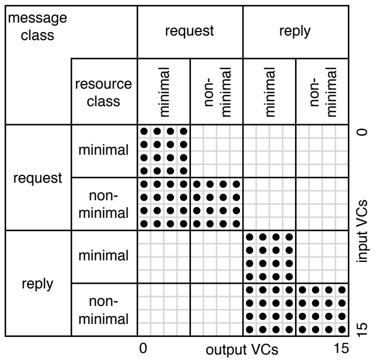


Figure 4: VC transition matrix (fbfly, $2\times2\times4$ VCs).

large wavefront block with $P \times V$ inputs can be replaced by M smaller wavefront blocks with $P \times V \times C$ inputs each.

Likewise, while a packet's resource class can change on its way through the network, it can by design only do so in a well-defined order. This allows us to further restrict the set of output VCs a given input VC can request; for example, for dateline routing, once a packet has crossed the dateline, it is no longer allowed to use the pre-dateline VCs. As a result, we can reduce the number of ports on the input- and output-side arbiters in the separable implementations down to the number of possible successor/predecessor resource classes times the number of VCs in each class. However, except for the special case where each resource class has at most one successor and one predecessor class (possibly including itself), this particular optimization does not apply to the wavefront-based implementation.

Finally, all VCs belonging to the same class are equivalent from a functional point of view. As a result, a given input VC can either use all of the output VCs in a given class or none of them, and thus VC allocator requests need not specify individual candidate VCs within the class, but instead can select the class as a whole. This allows for additional logic optimizations.

To demonstrate the sparseness of VC-to-VC transitions for a concrete example, Figure 4 shows all possible transitions for the case of the flattened butterfly topology with two message classes, two resource classes and four VCs assigned to each class. As indicated by the black circles, only 96 of the 256 total possible VC-to-VC transitions are actually legal—and thus need to be supported by the VC allocator—in this configuration; in particular, any given VC is restricted to at most eight possible successor and predecessor VCs, all of which are confined to the same matrix quadrant.

4.3 Results

4.3.1 Implementation Cost

Figure 5 shows the delay and area associated with each VC allocator implementation, while Figure 6 plots delay against power. For separable allocators, we consider both implementations built from matrix arbiters *(m)* and those using round-robin arbiters *(rr)*; since the pre-selection of a winning VC for each potentially granted output port is not on the wavefront allocator's critical path, we use the simpler round-robin arbiters for that case.

Connected data points in each of the subfigures represent designs before and after applying the optimizations described in Section 4.2; in cases where only a single data point is present, Design Compiler consistently ran out of memory while synthesizing the un-optimized version, and consequently only the optimized results are shown.

Overall, sparse VC allocation yields significant improvements across the board, reducing delay, area and power by up to 41%, 90% and 83%, respectively.

For the design points with a single VC assigned to each packet class, the sparse version of the wavefront allocator is both the fastest implementation and represents the best tradeoff between area and delay. However, as a result of its scaling properties, the wavefront allocator's delay quickly surpasses that of the separable implementations as the number of VCs increases; because synthesis tries to compensate for this effect by using faster—and therefore, larger—gates, the same applies for area and power.

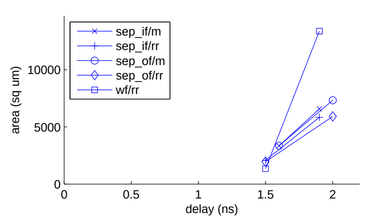
The difference in delay between separable implementations using matrix arbiters on the one hand and round-robin arbiters on the other hand is relatively small when sparse VC allocation is used; in most cases, the delay improvement is unlikely to justify the higher cost of using matrix arbiters.

Despite the optimizations proposed in Section 4.2, Design Compiler failed at synthesizing the wavefront-based allocator implementations for the two larger flattened butterfly configurations; however, extrapolating from the results for the comparatively simpler mesh configurations, their area, power and delay would significantly exceed those of the separable implementations at these design points. For the largest configuration of the flattened butterfly, synthesis could only be successfully completed for the two round-robin-based separable allocator variants.

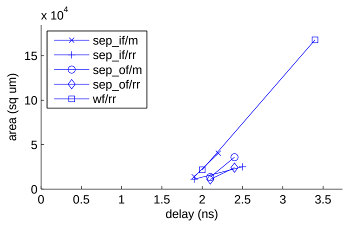
4.3.2 Matching Quality

Figure 7 shows the quality of matchings generated by each allocator. For cases where a single VC is assigned to each packet class, as shown in Figure 7(a) and 7(d), all three allocator implementations generate maximum matchings for every valid requests matrix, and thus have a constant matching quality—as defined in Section 3.1—of 1. In such cases, each input VC can use only one specific output VC; thus, if that VC is unavailable, there are no further candidate VCs it could use instead, and the request cannot be satisfied. Both the separable and the wavefront-based implementations will grant all non-conflicting requests and a single one among each group of conflicting requests, yielding the best possible matching under the given constraint.

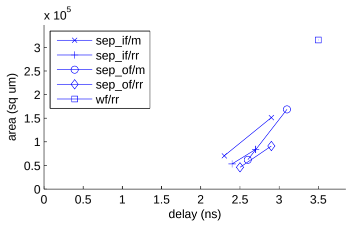
The situation is somewhat different in networks with multiple VCs assigned to each packet class. As before, all three allocator types are guaranteed to grant non-conflicting requests. However, in the presence of conflicts—i.e., multiple input VCs requesting access to the same packet class—the wavefront allocator will grant as many of the requests as VCs are available in that class, again yielding a maximum matching. For the separable allocators, on the other hand, situations can arise where some of the VCs within a given class are left unassigned even in the presence of further requests; e.g., for a separable input-first implementation, multiple input VCs destined for the same packet class might select the same output VC during input-side arbitration, leaving other VCs available in that class unused. Consequently, matching quality for the separable implementations decreases for both higher injection rates and larger numbers of VCs per class, as both increase the probability that such lockouts will



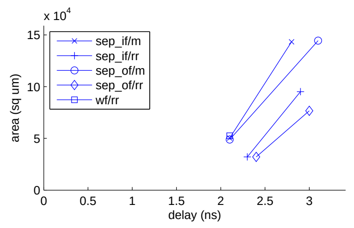
(a) mesh, 2×1×1 VCs.



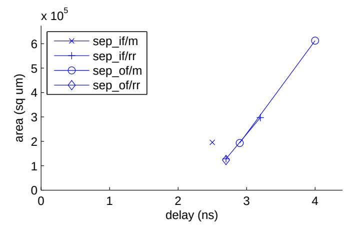
(b) mesh, 2×1×2 VCs.



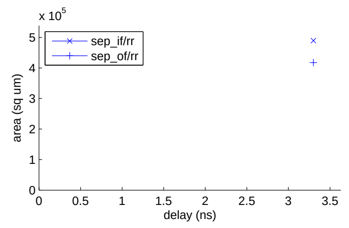
(c) mesh, 2×1×4 VCs.



(d) fbfly, 2×2×1 VCs.

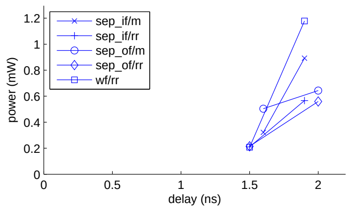


(e) fbfly, 2×2×2 VCs.

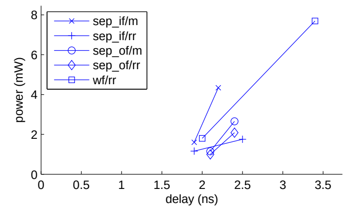


(f) fbfly, 2×2×4 VCs.

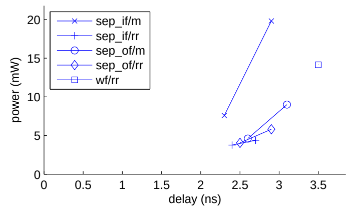
Figure 5: VC allocator area vs. delay.



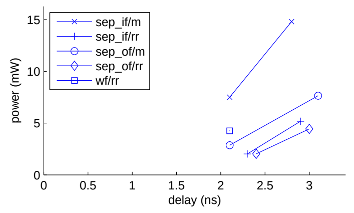
(a) mesh, 2×1×1 VCs.



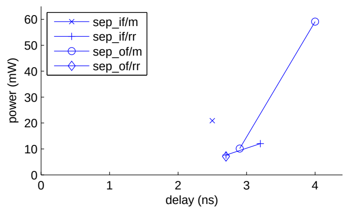
(b) mesh, 2×1×2 VCs.



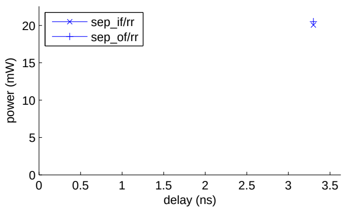
(c) mesh, 2×1×4 VCs.



(d) fbfly, 2×2×1 VCs.



(e) fbfly, 2×2×2 VCs.



(f) fbfly, 2×2×4 VCs.

Figure 6: VC allocator power vs. delay.

occur. Input-first allocation provides slightly better matching here, as the narrower V-input arbitration is performed before the wider $P{\times}V$-input arbitration, resulting in more requests being propagated from the first stage of arbitration to the second one than in the output-first case.

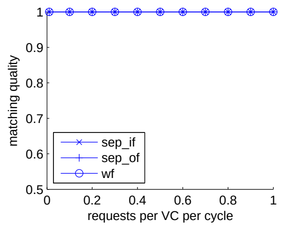
Overall, we find that a wavefront-based VC allocator yields a matching quality of 1 for all configurations, and outperforms separable input- and output-first implementations by up to 20% and 25%, respectively, under high load.

4.3.3 Network-Level Performance

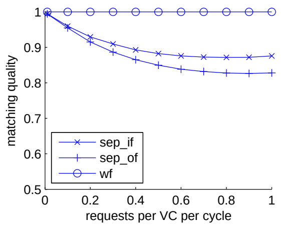
Due to the fact that VC allocation only needs to be performed once per packet, the impact on overall network performance is less pronounced than the differences in matching quality would imply: As outlined in Section 3, a request-reply packet pair in our traffic model always comprises a total of six flits; consequently, on average only one in three flits requires VC allocation, making it unlikely for multiple conflicting VC requests to be issued in the same cycle at any given router. As non-conflicting requests are guaranteed to be granted by all three allocator implementations considered, we expect VC allocation quality to have little overall impact on network performance. Indeed, network-level simulation results—not presented in more detail due to space constraints—confirm that the choice of VC allocator does not significantly affect the latency-throughput characteristics for otherwise identical router configurations; in particular, both zero-load latency and saturation bandwidth remain virtually unchanged.

4.4 Discussion

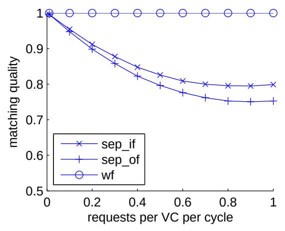
Overall, the VC allocator's limited impact on network-level performance suggests that architects can essentially ignore matching quality when selecting a particular imple-



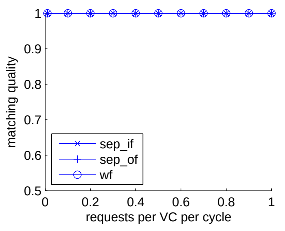
(a) mesh, 2×1×1 VCs.



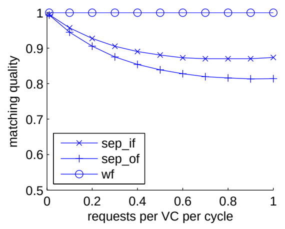
(b) mesh, 2×1×2 VCs.



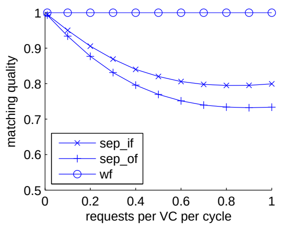
(c) mesh, 2×1×4 VCs.



(d) fbfly, 2×2×1 VCs.



(e) fbfly, 2×2×2 VCs.



(f) fbfly, 2×2×4 VCs.

Figure 7: VC allocator matching quality.

mentation, and choose the allocator architecture that best matches given delay, area and power constraints.

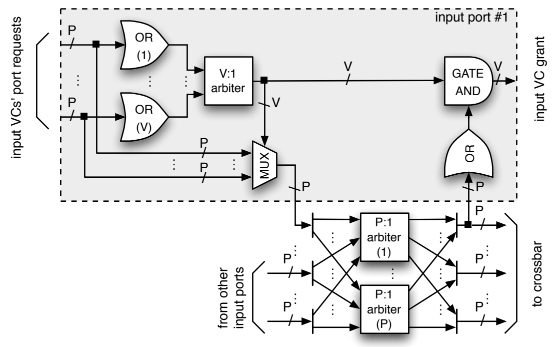
While previous work in the NoC domain has largely employed separable allocator implementations throughout, our results indicate that, after applying the optimizations introduced in Section 4.2, wavefront-based VC allocator implementations represent a reasonable design choice for networks with few VCs assigned to each message class. As the number of VCs increases, though, the wavefront allocator's delay and cost grow rapidly, particularly for the flattened butterfly configurations; in this case, the separable input-first allocator provides low delay, as well as reasonable matching quality and cost, and consequently represents a more suitable choice.

5. SWITCH ALLOCATORS

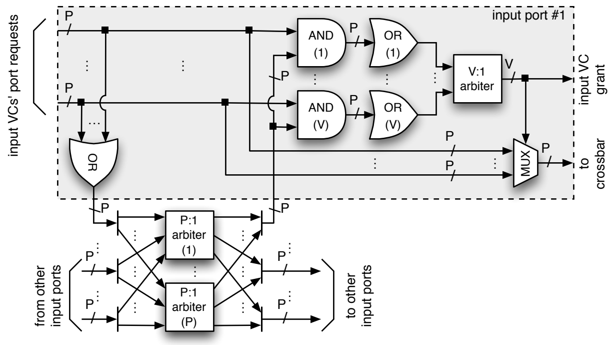
In this section, we apply the evaluation methodology used in the previous section to three representative switch allocator implementations and propose an improved speculation mechanism that reduces critical path delay without incurring an increase in zero-load latency.

5.1 Implementation Details

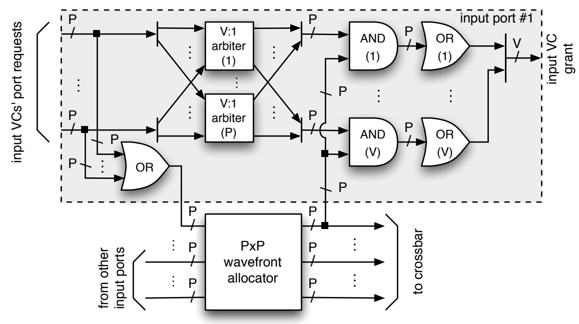
Switch allocation performs a matching between requests from the V input VCs at each of the router's P input ports on the one hand and available crossbar slots on the other hand, subject to the constraint that at most one VC per input port can receive a grant. As a result of this additional



(a) Separable input-first allocator (sep_if).



(b) Separable output-first allocator (sep_of).



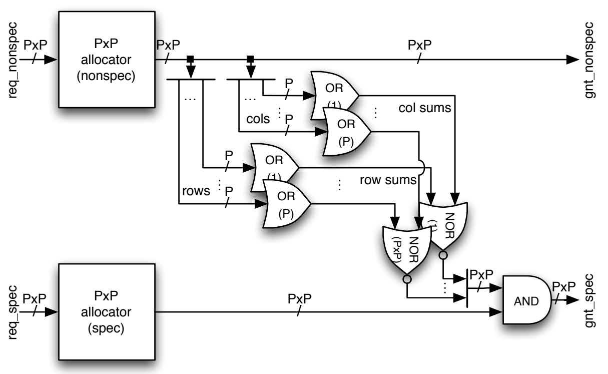
(c) Wavefront allocator (wf).

Figure 8: Switch allocator block diagrams.

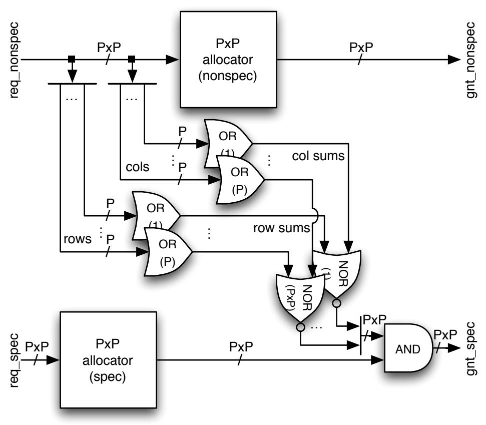
constraint, switch allocators differ slightly from canonical $P \times V$-input allocators; block diagrams for the switch allocator implementations considered in this paper are shown in Figure 8.

In the separable input-first implementation, shown in Figure 8(a), a V-input arbiter first determines a winner among all active VCs at each input port. The winning VCs' requests are then forwarded to the appropriate output ports, where a second round of P-input arbitration takes place among requests from different inputs. Grants are generated for those VCs whose requests are successful in both arbitration stages. Furthermore, the output arbiters directly drive the control signals to the crossbar.

In the output-first implementation, shown in Figure 8(b), requests from all input VCs are combined and forwarded to the appropriate output ports, where P-input arbitration is



(a) Conventional scheme.



(b) Pessimistic scheme.

Figure 9: Speculative switch allocation.

performed between all requesting input ports. Once one or more output ports are granted to a given input port, we check which of its input VCs can use the granted ports, and perform V-input arbitration among all candidates to determine a winning VC. Because multiple outputs might be granted to a given input in the first stage, the crossbar control signals in this case cannot be driven directly by the output-stage arbiters; instead, they are generated from the winning VC's port select signal after allocation is complete.

Finally, for the wavefront allocator, shown in Figure 8(c), different input VCs' requests are combined as in the output-first case; however, the wavefront block guarantees that at most one output port is granted to any given input port. As a result, the wavefront block's outputs can directly drive the crossbar control signals, and overall delay can be reduced by pre-determining for each input port which of its VCs will be selected if a grant for a given output port is received; the pre-selection can be performed in parallel with the wavefront allocation by a stage of V-input arbiters.

5.2 Speculative Switch Allocation

Speculative switch allocation as a means of reducing a router's zero-load latency was originally proposed by Peh and Dally [17]. It enables head flits to effectively bypass the VC allocation stage in the router pipeline by allowing them to bid for crossbar access at the same time they request an output VC.

The implementation described in their original paper uses separate switch allocators for handling non-speculative and speculative requests. In order to avoid performance degradation due to misspeculation, non-speculative grants are prioritized over speculative ones; i.e., speculative grants are discarded if any non-speculative grant that uses the same input or output port is generated. A block diagram of a possible implementation is shown in Figure 9(a). The checks for input and output conflicts require a set of $2\times P$ P-input reduction ORs to generate row- and column-wise summary bits indicating the presence of a non-speculative grant, followed by a two-input NOR stage and a two-input AND stage to perform the actual masking. Assuming the same basic allocator implementation is used for speculative and non-speculative requests, the allocator's overall critical path is thus extended by the sum of the delays corresponding to the reduction trees and the masking logic when compared to a non-speculative implementation.

However, we can reduce this additional delay without significantly impacting speculation efficiency by observing that speculation has the most noticeable effect at low to medium network load, where the expected number of requests pending at each router's inputs—and thus the probability of multiple requests being in conflict—is low. Consequently, it is likely that the majority of requests will be granted, and we can pessimistically mask speculative grants based on conflicting non-speculative requests—rather than non-speculative grants—without sacrificing a significant fraction of the available speculation opportunities. As shown in Figure 9(b), this removes both the large reduction ORs and the NOR gates from the overall critical path, reducing the delay overhead over a non-speculative implementation to the final stage of 2-input AND gates.

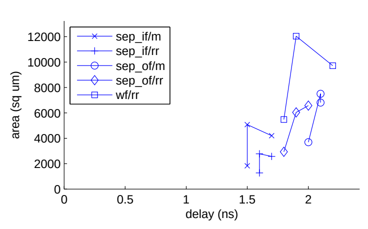
5.3 Results

5.3.1 Implementation Cost

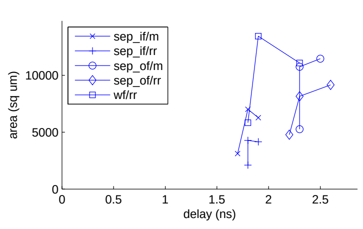
Figures 10 and 11 show the delay-area and delay-power tradeoffs for the three switch allocator implementations described in Section 5.1. The three data points on each curve correspond to a non-speculative implementation, a pessimistic speculative implementation and a conventional speculative implementation, respectively.

Overall, the separable input-first allocator consistently offers the lowest delay and in most cases pareto-dominates both the corresponding output-first and wavefront configurations. The wavefront-based implementations approach the delay of the input-first ones for a number of mesh design points, but more generally fall between the latter and the output-first configurations, while being more expensive than either in terms of area and power. As in the case of the VC allocators, the delay reduction achieved by using matrix arbiters rather than simpler round-robin arbiters for the separable implementations usually does not outweigh the associated area and power penalty.

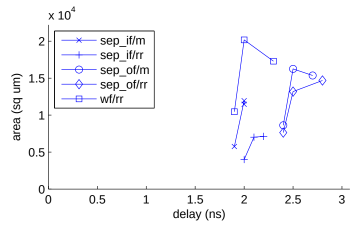
As expected, the pessimistic speculation mechanism yields lower delay than the conventional one virtually across the board, in many cases approaching that of a non-speculative implementation. The delay reduction is most pronounced for the wavefront allocator, where savings of up to 23% can be achieved, but is usually accompanied by increases in area and power in this context. For some of the separable configurations, on the other hand, area and power are reduced



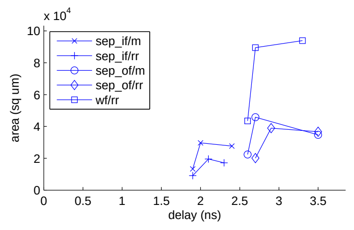
(a) mesh, 2×1×1 VCs.



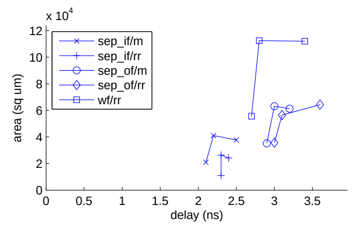
(b) mesh, 2×1×2 VCs.



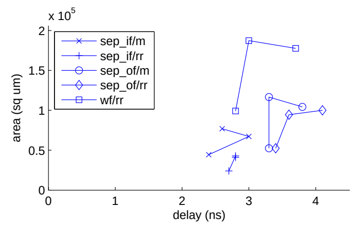
(c) mesh, 2×1×4 VCs.



(d) fbfly, 2×2×1 VCs.

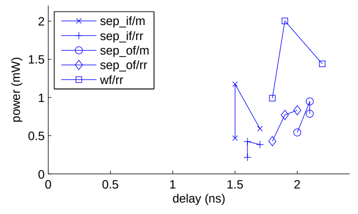


(e) fbfly, 2×2×2 VCs.

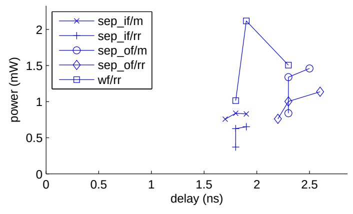


(f) fbfly, 2×2×4 VCs.

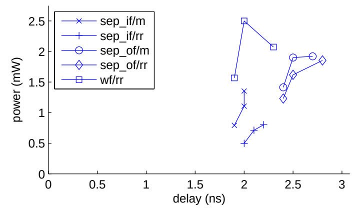
Figure 10: Switch allocator area.



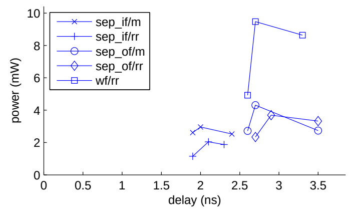
(a) mesh, 2×1×1 VCs.



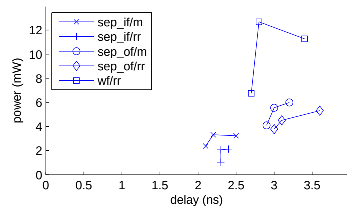
(b) mesh, 2×1×2 VCs.



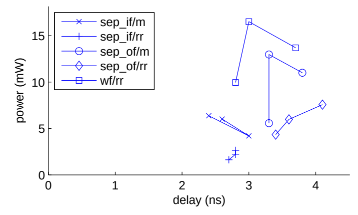
(c) mesh, 2×1×4 VCs.



(d) fbfly, 2×2×1 VCs.



(e) fbfly, 2×2×2 VCs.



(f) fbfly, 2×2×4 VCs.

Figure 11: Switch allocator power.

along with delay.

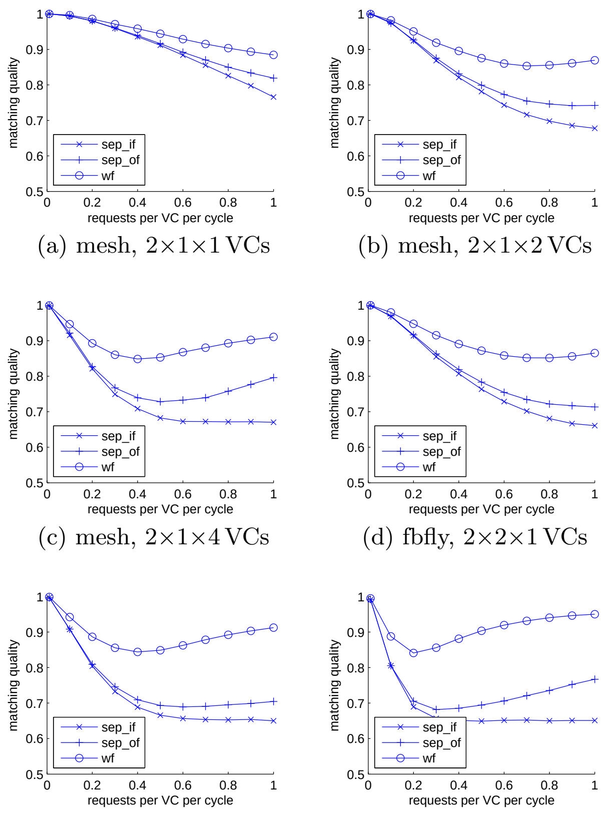
5.3.2 Matching Quality

Figure 12 compares the individual switch allocators' matching quality. At low network loads, all three allocators generate near-maximum matchings; as in the case of the VC allocator, this is because conflicting requests are less likely to occur at low injection rates.

As the injection rate is increased, the number of matchings generated by the wavefront-based allocator initially ramps up more slowly than the one generated by a maximum-size allocator, and its matching quality thus starts to decrease. However, for network configurations with larger numbers of VCs and at high injection rates, the probability that at least one VC at each input port has a flit available for each output port in any given cycle eventually becomes high. This presents a natural limit to the number of matchings a maximum-size allocator—and, by extension, any allocator—can generate. As the wavefront allocator is still progressing towards this limit even once the maximum-size allocator has reached it, its matching quality starts to increase again.

The separable output-first allocator, on the other hand, is not guaranteed to find maximal matchings. While its matching quality is therefore inferior to that of the wavefront-based implementation, its general behavior is otherwise similar.

In contrast, the separable input-first switch allocator eventually becomes limited by the fact that it can only propagate a single request per input port to its second arbitration stage; consequently, it's matching quality starts to flatten out once the probability of at least one VC at a given input port having a flit available in any given cycle becomes high, and is generally lower than the one offered by the other two allocator implementations.



(a) mesh, 2×1×1 VCs (b) mesh, 2×1×2 VCs (c) mesh, 2×1×4 VCs (d) fbfly, 2×2×1 VCs (e) fbfly, 2×2×2 VCs (f) fbfly, 2×2×4 VCs

Figure 12: Switch allocator matching quality.

5.3.3 Network-Level Performance

The average network-level packet latency as a function of the flit injection rate is depicted in Figure 13. Based on our conclusions from Section 4, simulations assume a separable input-first VC allocator; furthermore, we use the improved speculative switch allocation mechanism as described in Section 5.1.

Despite the differences in matching quality, we find that input- and output-first separable switch allocators yield virtually identical average packet latencies across the entire range of injection rates and network configurations. This discrepancy results from the fact that the isolated simulations conducted for measuring matching quality generate requests for each input VC independently, which can lead to much higher request rates than could be sustained in steady state in an actual network.

Comparing the wavefront allocator and the separable allocators, the difference in saturation rate is negligible for the mesh topology with 2×1×1 VCs and remains below 4% for 2×1×4 VCs. For the flattened butterfly, on the other hand, the performance advantage is more significant, particularly as the number of VCs is increased: For 2×2×1 VCs, the wavefront allocator improves the saturation rate by 4% compared to the separable input-first allocator; for 2×2×4 VCs, the difference exceeds 20%. The wavefront allocator benefits both from larger (i.e., higher radix) and from more densely populated (i.e., more VCs) request matrices, as these provide more candidate assignments for finding a good maximal matching. For separable allocators, on the other hand, such request matrices actually increase the likelihood of multiple output ports being assigned to the same input or vice versa, due to the fact that input and output arbitration are performed separately.

Figure 14 compares the average packet latency achieved by the two speculative switch allocation mechanisms described in Section 5.2, as well as by a non-speculative switch allocator. Simulations were performed using a separable input-first switch allocator; additional simulations using separable output-first and wavefront allocators yielded similar results; for the wavefront allocator, the differences in saturation rate were slightly less apparent.

The impact of speculation on the network's zero-load latency is more pronounced for the mesh network, where router pipeline delay represents a larger fraction of overall packet latency and we measure improvements of up to 23%. Due to the flattened butterfly's much smaller network diameter, zero-load latency for this topology is dominated by channel and serialization latency and as a result is only improved by 14%.

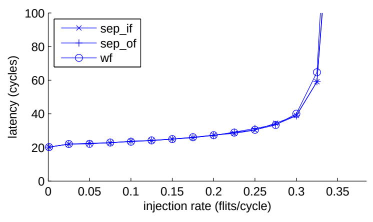
Speculative switch allocation can also help increase the network's saturation rate, as it avoids stall cycles caused by flits waiting for VC allocation and as a result allows more flits to traverse the router per unit time. However, in networks with sufficiently many VCs, there is a high probability that a stall cycle can be avoided by just issuing a flit from another VC instead. Consequently, the saturation rate improvement due to speculation is larger in networks with fewer VCs. We measure improvements of 14% for the mesh network with 2×1×1 VCs, 6% for the flattened butterfly with 2×2×1 VCs, and less than 5% for the remaining configurations of either topology.

As expected, both speculative implementation variants yield virtually identical performance for low to medium injection rates where the majority of requests are granted. As the injection rate approaches saturation and the probability of conflicts increases, the pessimistic variant discards a larger fraction of speculation opportunities, and as a result, its latency approaches that of the non-speculative implementation. This effect is slightly more pronounced for networks with larger numbers of VCs, where statistically more requests are available at each input port, and thus conflicts are more likely to occur.

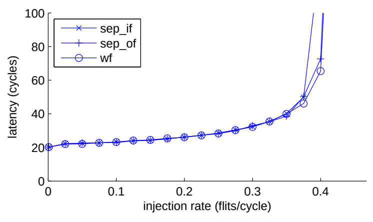
Overall, our results indicate that the pessimistic approach of suppressing speculative grants by checking for conflicting non-speculative requests can reduce delay compared to the conventional approach without compromising zero-load latency. While it becomes less effective once the network starts to approach saturation, the maximum throughput is only reduced by less than 4% compared to the conventional speculation mechanism.

5.4 Discussion

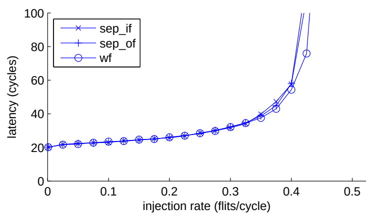
From an architect's point of view, switch allocators that provide higher matching quality at the cost of increased delay are particularly suitable for improving performance in primarily throughput-oriented networks, where large quantities of data are transferred concurrently using DMA-like semantics. Examples of such networks include I/O interfaces connecting different functional blocks on a SoC, or the data supply networks for highly parallel graphics accelerators and stream computing engines. For primarily latency-sensitive



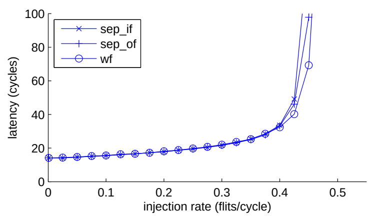
(a) mesh, 2×1×1 VCs



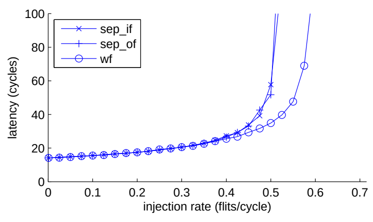
(b) mesh, 2×1×2 VCs



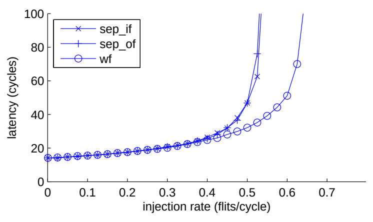
(c) mesh, 2×1×4 VCs



(d) fbfly, 2×2×1 VCs

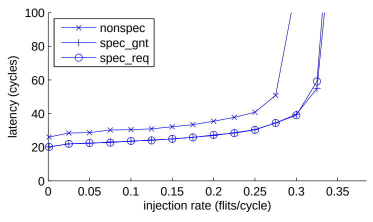


(e) fbfly, 2×2×2 VCs

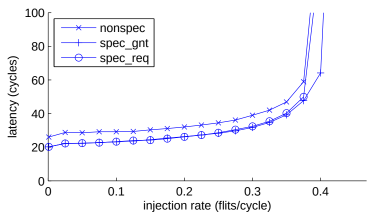


(f) fbfly, 2×2×4 VCs

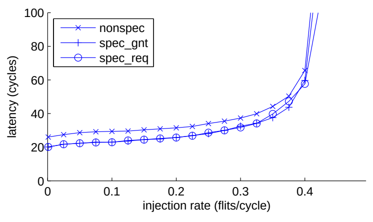
Figure 13: Performance of different switch allocator implementations.



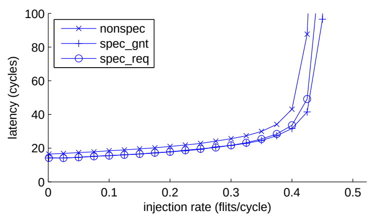
(a) mesh, 2×1×1 VCs



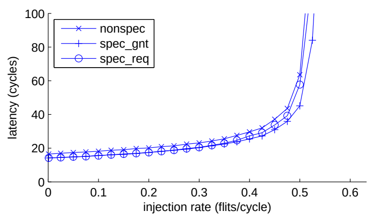
(b) mesh, 2×1×2 VCs



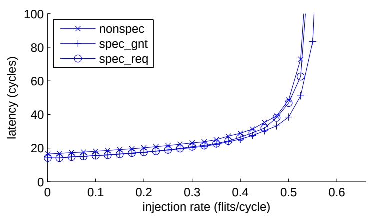
(c) mesh, 2×1×4 VCs



(d) fbfly, 2×2×1 VCs



(e) fbfly, 2×2×2 VCs



(f) fbfly, 2×2×4 VCs

Figure 14: Performance of speculative switch allocation schemes.

applications like cache coherence traffic, on the other hand, network load is expected to be relatively low during normal operation; such applications favor separable allocators due to their comparatively smaller delay and cost.

Speculation, on the other hand, is most useful in latency-sensitive applications that directly benefit from the resulting decrease in zero-load latency. For throughput-oriented networks, the slight increase in saturation rate afforded by speculation is unlikely to justify the associated increases in delay and complexity. Furthermore, speculation is less attractive for topologies with low network diameter, where pipeline delay accounts for only a small fraction of the overall packet latency.

6. CONCLUSIONS

In this paper, we have explored the design space for VC and switch allocators in the context of NoC routers. In particular, we have evaluated input- and output-first variants of separable allocators as well as wavefront allocators, and characterized them in terms of matching quality, delay, area and power.

Simulation results for two representative 64-node NoC topologies indicate that despite measurable differences in matching quality, network-level performance is largely insensitive to the choice of VC allocator. As such, a particular implementation can be selected primarily on the basis of delay and cost considerations without sacrificing performance. While the wavefront allocator represents a reasonable choice for smaller networks, separable variants offer lower delay and cost for networks with higher radix and more VCs.

In contrast, the choice of switch allocator directly affects the achievable saturation rate, particularly for larger networks. While the difference compared to separable allocators is small for mesh networks with few VCs, the wavefront allocator achieves 15% and 21% more throughput for a flattened butterfly network with 8 and 16 VCs, respectively. On the other hand, separable input-first allocators tend to offer lower delay, area and power.

We have also introduced a sparse VC allocation scheme, which can reduce the VC allocator's delay, area and power by up to 41%, 90% and 83%, respectively. Finally, we have shown that by using a slightly pessimistic approach to speculative switch allocation, the delay of the switch allocator can be reduced by up to 23% without compromising zero-load latency.

Overall, our results indicate that choosing the optimal allocator implementation depends on the expected network and application characteristics and constraints; in particular, different tradeoff points are favored by latency-sensitive and throughput-oriented applications, respectively.

7. ACKNOWLEDGMENTS

This research was supported in part by the National Security Agency under Contract H98230-08-C-0272-P-3 and the National Science Foundation under Grant CCF-0702341. Daniel U. Becker is supported by a Prof. Michael J. Flynn Stanford Graduate Fellowship.

8. REFERENCES

[1] L. Benini and G. de Micheli. Networks on Chip: A New Paradigm for Systems on Chip Design. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, 2002.

[2] W. J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2), 1992.

[3] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Inteconnection Networks. In *Proceedings of the 38th Conference on Design Automation (DAC-38)*, 2001.

[4] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[5] J. G. Delgado-Frias and G. B. Ratanpal. A VLSI Wrapped Wave Front Arbiter for Crossbar Switches. In *Proceedings of the 11th Great Lakes Symposium on VLSI*, 2001.

[6] L. R. Ford and D. R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8(3), 1956.

[7] M. Galles. Spider: A High-Speed Network Interconnect. *IEEE Micro*, 17(1), 1997.

[8] R. R. Hoare, Z. Ding, and A. K. Jones. A Near-optimal Real-time Hardware Scheduler for Large Cardinality Crossbar Switches. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'06)*, 2006.

[9] J. Hurt, A. May, X. Zhu, and B. Lin. Design and Implementation of High-Speed Symmetric Crossbar Schedulers. In *Proceedings of the 1999 IEEE Conference on Communications (ICC'99)*, volume 3, 1999.

[10] J. Kim, J. Balfour, and W. J. Dally. Flattened Butterfly Topology for On-Chip Networks. In *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture (ISCA-40)*, 2007.

[11] J. Kim, C. Nicopoulos, D. Park, N. Vijaykrishnan, Y. S. Mazin, and C. R. Das. A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks. *SIGARCH Computer Architecture News*, 34(2), 2006.

[12] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. K. Jha. A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS. In *Proceedings of the 2007 IEEE International Conference on Computer Design (ICCD'07)*, 2007.

[13] N. McKeown. The iSLIP Scheduling Algorithm for Input-Queued Switches. *IEEE/ACM Transactions on Networking*, 7(2), 1999.

[14] S. S. Mukherjee, F. Silla, P. Bannon, J. S. Emer, S. Lang, and D. Webb. A Comparative Study of Arbitration Algorithms for the Alpha 21364 Pipelined Router. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002.

[15] R. Mullins, A. West, and S. Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *Proceedings of the 31st International Symposium on Computer Architecture (ISCA-31)*, 2004.

[16] D. Park, R. Das, C. Nicopoulos, J. Kim, N. Vijaykrishnan, R. K. Iyer, and C. R. Das. Design of a Dynamic Priority-Based Fast Path Architecture for On-Chip Interconnects. In *Proceedings of the 15th Symposium on High Performance Interconnects (HOTI-15)*, 2007.

[17] L.-S. Peh and W. J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA-7)*, Apr 2001.

[18] A. Singh. *Load-Balanced Routing in Interconnection Networks*. PhD thesis, Stanford University, 2005.

[19] Y. Tamir and H.-C. Chi. Symmetric Crossbar Arbiters for VLSI Communication Switches. *IEEE Transactions on Parallel and Distributed Systems*, 4(1), 1993.

[20] L. G. Valiant and G. J. Brebner. Universal Schemes for Parallel Communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, 1981.