Stay in your Lane: A NoC with Low-overhead Multi-packet Bypassing

Hossein Farrokhbakht*, Paul V. Gratz†, Tushar Krishna‡, Joshua San Miguel§, Natalie Enright Jerger*
*University of Toronto, †Texas A & M, ‡Georgia Institute of Technology, §University of Wisconsin-Madison
Email: *h.farrokhbakht@mail.utoronto.ca, †pgratz@tamu.edu, ‡tushar@ece.gatech.edu,
§jsanmiguel@wisc.edu, *enright@ece.utoronto.ca

***Abstract*—NoCs are over-provisioned with large virtual channels to provide deadlock freedom and performance improvement. This use of virtual channels leads to considerable power and area overhead. In this paper, we introduce a novel flow control, called FastFlow, to enhance performance and avoid both protocol- and network-level deadlocks with an impressive reduction in number of virtual channels compared to the state-of-the-art NoCs. FastPass promotes a packet to traverse the network bufferlessly; the packet bypasses the routers to reach its destination. During the traversals, the packet is guaranteed to make forward progress every cycle. As a result, such a packet cannot be blocked by congestion nor deadlock. Promoting more packets to FastPass will provide higher throughput. To this end, FastPass allows multiple packets to be upgraded as FastPass packets simultaneously. To avoid any collision between these packets, FastPass provides multiple pre-defined non-overlapping lanes. Each lane is allowed to propagate only one FastPass packet. In a time-division multiplexed way, each router gets a chance to upgrade its packets to the FastPass packets and then transfer them via the pre-defined non-overlapping lanes. FastPass not only provides high throughput but also resolves both protocol- and network-level deadlocks. Compared to the state-of-the-art, FastPass presents a 1.8× increase in throughput for synthetic traffic, 46% improvement in average packet latency for real applications, and 40% reduction in power and area consumption.**

I. Introduction

Efficient communication in many-core processors is critical to overall performance. Latency is traditionally reduced by avoiding router pipeline stages through bypassing. Throughput is traditionally improved by increasing buffering and virtual channels (VCs) to reduce head of line blocking. Efficiently bypassing multiple routers often breeds complexity. Increasing buffering contributes non-negligibly to router area and power consumption. In this paper, we propose a time-division-multiplexed (TDM) approach to enabling bypass paths that improves both latency and throughput while reducing the overall buffering of the router compared to the baseline.

Until recently, conventional routers had a large minimum number of buffers that they had to support. The minimum number of buffers is typically driven by correctness requirements imposed by the coherence protocol; messages from different classes must be separated into different virtual networks (composed of buffers) to avoid protocol-level deadlock. Recent work [13], [24] demonstrates alternative deadlock-freedom solutions that eliminate the need for virtual networks (VNs). However, eliminating VNs entirely hurts throughput. *Can we design a new flow control mechanism such that we reduce buffering requirements while improving performance and also providing deadlock freedom?*

Flow control optimizations have been proposed to improve throughput and latency [19], [20]. Circuit switching is a form of flow control that bypasses the router pipeline [11]. These flow control techniques often require the exchange of credits or handshaking to ensure that a path is available. The Runahead NoC [21] proposes a second, lightweight network that speculatively delivers packets but will drop them mid-transmission if there is a conflict. Rather than pay the area cost of an extra network, we propose to use one network to deliver fast and regular packets. By carefully designing an algorithm that governs which packets are locally selected by which routers and the path they must follow, we can reduce latency and improve throughput.

The time-division multiplexed, non-overlapping paths must be determined a priori. As NoCs scale to increasing numbers of cores, attempting to coordinate communication across multiple routers becomes undesirable. This problem is well known in adaptive routing; obtaining up-to-date congestion information is challenging and has motivated region-based approaches [15], [22]. Similarly in flow control, coordination such as to establish a circuit-switched path increases latency by requiring a setup phase. To avoid any global coordination or setup phase, our design, called `FastPass`, has a predetermined algorithm to select which router can promote a packet to a fast path. The algorithm moves from router to router in a time-division multiplexed way. Dynamically dependent on the load, packets reach their destinations as fast or regular packets. But only allowing one router to use a fast path at a time would not help performance as the network scales. Thus, we propose a novel scheme to partition the network such that non-overlapping bypass paths are enabled at each step for multiple routers. This allows us to provide significant throughput enhancement and deadlock freedom at the same time.

This paper makes the following primary contributions:

- Proposes a novel flow control, FastFlow to route packets bufferlessly across non-overlapping paths; as the load increases, FastFlow kicks in so that the promoted packets can avoid congested areas.

- Proves that our FastPass also enables deadlock freedom by promoting blocked packets; Table I provides a comparison between FastPass and other deadlock freedom techniques.
- Shows that high throughput can be achieved with low complexity and low buffering requirements.
- Compared to the state-of-the-art, FastPass provides a 1.8× increase in throughput for synthetic traffic, 46% improvement in average packet latency and 9% reduction in execution time for real applications, and 40% reduction in power and area consumption.

II. Background

Deadlock Freedom: Providing correctness is of utmost importance in a multiprocessor interconnection network. A NoC design guarantees a cornerstone of correctness by being free of deadlock. A deadlock happens when a set of agents sharing a set of resources wait indefinitely to acquire the resources in a cyclic dependence. Within a NoC, packets are the set of agents while buffers are the set of resources storing incoming packets as they move in the NoC to reach their destinations. In the interconnection network, two types of deadlocks may emerge: protocol- and network-level deadlocks. In both cases, packets stall indefinitely causing a system failure. Maintaining deadlock-free communication imposes overheads in the form of VCs, performance issues and/or hardware complexity.

Implementing cache coherence on the NoC causes atomic coherence transactions to be divided into several non-atomic packets. If these packets interleave with each other while sharing the same buffers, **protocol-level deadlock** can happen. For example, if two cache controllers issue a burst of coherence requests using up all of the buffers in the network, both processors will be stalled awaiting responses; thus they stop processing requests packets. As responses use the same buffers as requests, the remote responses cannot make forward progress leading to a protocol-level deadlock. A **network-level deadlock** happens when in a cyclic dependence chain, multiple packets occupy buffers while awaiting other occupied buffers to become free. Since packets do not give up the buffers they are residing in, they will wait forever. This situation happens when adaptive routing is employed; adaptive routing allows cycles to form since packets can be routed in any directions.

III. FastPass

Analogy: As an analogy for FastPass, consider a Disneyland park; to enjoy the popular rides, people need to wait in a long line. However, Disneyland provides *FastPass* service by which people can skip the long line minimizing the amount of waiting time and maximizing the number of rides. In a given fixed time slot, people may use the FastPass for a few rides. In the next time slot, they may use the FastPass service again for other rides. During each time slot, taking the rides through the *regular pass* is also available. Similarly, FastPass provides recurrent fixed time slots. In a given time slot, only a subset of routers may enable FastPass service simultaneously. When the time slot elapses, another set of routers provides FastPass service. Each FastPass service guarantees a single packet reaches its destination whether the downstream router has enough space or not.

High-level idea: To provide throughput enhancement and deadlock-free communication, we use a predetermined algorithm enabling FastPass service for packets. *The FastPass service guarantees that packets make forward progress every cycle.* Each router may use the FastPass service to promote its packets. The upgraded packets then traverse the FastPass path bufferlessly to reach the destinations. To maximize the benefit of the FastPass service, we let a subset of routers use the FastPass service at the same time by providing recurring fixed time slots. To avoid collision between the FastPass packets, we use pre-defined non-overlapping bypass paths which cover all routers in the network. In a given time slot, each router using FastPass service has access to some pre-defined paths over which the FastPass packets traverse bufferlessly to their destinations. In the next time slot, the routers have access to another set of pre-defined paths over which the packets can reach their destinations. This process continues until the routers have access to all destinations in the network. Once a set of routers complete this procedure, in a statically predetermined order, another set of routers may enable FastPass service. In turn, all routers in the network will have a chance to upgrade packets to FastPass packets. The beauty of this approach is that it provides significant throughput enhancement and resolves both protocol- and network-level deadlocks. This is because that the packets traversing the NoC via the FastPass service are guaranteed to make forward progress every cycle.

A. Definitions

We use following terminology:

- **Partition:** A partition includes a set of routers. There are multiple independent partitions in the network.
- **FastPass-Lane:** A path which covers one partition in the network. At any given time, there are multiple non-overlapping FastPass-Lanes in the network.
- **Prime Router:** Prime routers are the set of routers (one per partition) that are currently allowed to use the FastPass-Lanes. Each FastPass-Lane provides an isolated path from each prime router to the routers of a single partition.
- **FastPass-Packet:** A packet picked by the prime router to traverse bufferlessly. At any given time, there is only one FastPass-Packet traversing through a FastPass-Lane. Each prime router selects a FastPass-Packet whose destination lies in the currently covered partition.

Table I: Comparison of different deadlock freedom solutions.

Proposed Solutions	No Detection	**Protocol Deadlock Freedom**	Network Deadlock Freedom	Full Path Diversity	**High-throughput**	**Low-power**	**Scalability**	**No Misrouting**
Turn Restrictions [7]	✓	✗	✓	✗	✗	✗*	✗****	✓
Escape VCs [8]–[10], [14]	✓	✗	✓	✗**	✗	✗*	✓	✓
Virtual Networks [23]	✓	✓	✗	✗	✗	✗*	✓	✓
SPIN [31]	✗	✗	✓	✓	✗	✗*	✗*****	✓
SWAP [26]	✓	✗	✓	✓	✗	✗*	✓	✗
DRAIN [24]	✓	✓	✓	✓	✗	✗***	✗*****	✗
Pitstop [13]	✓	✓	✓	✓	✗	✓	✗*****	✓
Our Method: FastPass	✓	✓	✓	✓	✓	✓	✓	✓

* Must use multiple VNs to avoid protocol-level deadlock.
** No full path diversity within the escape VC.
*** DRAIN can work with no VNs; however, it requires a large amount of buffer which is non-minimal [13].
**** Cannot support adaptive routing.
***** As the network size increases, the time for detecting/resolving deadlock increases.

- **FastFlow:** A new flow control scheme allowing FastPass-Packets to make forward progress bufferlessly every cycle regardless of credit availability.
- **Regular Pass:** A credit-based flow control which is used when packets do not use the FastFlow control.
- **Regular Packets:** Packets using regular pass to make forward progress.

B. FastFlow

The goal of this work is to design a high-throughput and low-cost solution to minimize the head of line blocking and provide network- and protocol-level deadlock freedom in NoCs. To this end, we propose an efficient flow control called FastFlow. Packets are routed using either FastFlow or *regular pass* flow (i.e., credit-based) control. A FastPass-Packet traversing via FastFlow takes precedence over regular packets. A FastPass-Packet reaches its destination bufferlessly using a minimal path; no misrouting is employed. As a result, buffer turnaround time is eliminated increasing throughput. To maximize throughput, `FastPass` allows multiple FastPass-Packets to traverse the network at any given time. However, to guarantee no collisions between FastPass-Packets, `FastPass` creates multiple non-overlapping paths (FastPass-Lanes) allowing a single router access to each partition in a time-multiplexed way. Thus links of each partition only propagate one FastPass-Packet in a given slot time. Section III-C1 demonstrates how `FastPass` guarantees multiple FastPass-Packets with non-overlapping paths.

C. Detailed Algorithm

1) TDM-based Non-overlapping FastPass-Lanes: `FastPass` allows multiple FastPass-Packets to traverse the network simultaneously to maximize throughput. However, since the FastPass-Packets traverse bufferlessly, they must not collide with each other. To guarantee no collision between FastPass-Packets, `FastPass` creates multiple pre-defined non-overlapping paths in the network, called *FastPass-Lanes*. This problem is similar to maintaining non-interference in NoCs. Non-interference means that packets from one application should have no effect on the delivery of packets from other applications [36]. Here prime routers are equivalent to the applications. The goal is to implement such a non-interference approach that guarantees no collision between FastPass-Packets.

To this end, `FastPass` breaks the topology into P separate partitions; there is one prime router in each partition. This spatial partitioning provides the foundation for our collision-free FastPass-Packet traversals. Assuming a mesh topology, P could be the number of columns/rows. An efficient way to support multiple FastPass-Lanes with no overlap is by time-multiplexing the physical links. `FastPass` provides *recurrent fixed time slots* during which the links of each partition are only permitted to propagate a single packet from one prime router. When the time slot elapses, the links of each partition handle the FastPass-Packets of a different prime. This rotation continues until each prime router has a FastPass-Lane with every other router. The time needed for a such coverage is called a *phase*. Each phase consists of multiple time slots. In other words, there exists a fixed time slice for each phase where every prime router would have the opportunity to have a FastFlow with every other router in the network. In the next phase, the next set of prime routers follow the same procedure. Note that all routers along a single partition are *possible* destinations so the number of FastPass-Lanes a prime router has to rotate through is very limited and scales gracefully.

This time-multiplexing scheme guarantees that the path used by each prime router is completely independent of the other primes' paths. As a result, the packets of each prime cannot collide with others providing collision-free traversal. As an example, Fig. 1 demonstrates the non-overlapping FastPass-Lanes in a 3×3 mesh network.

The prime router within each partition is chosen contiguously meaning that the prime ability is given to the next adjacent router within the partition after finishing each phase. If the current prime is located in the last row, the router in the first row of the partition will be the next prime router. Note that the current prime router in the last row

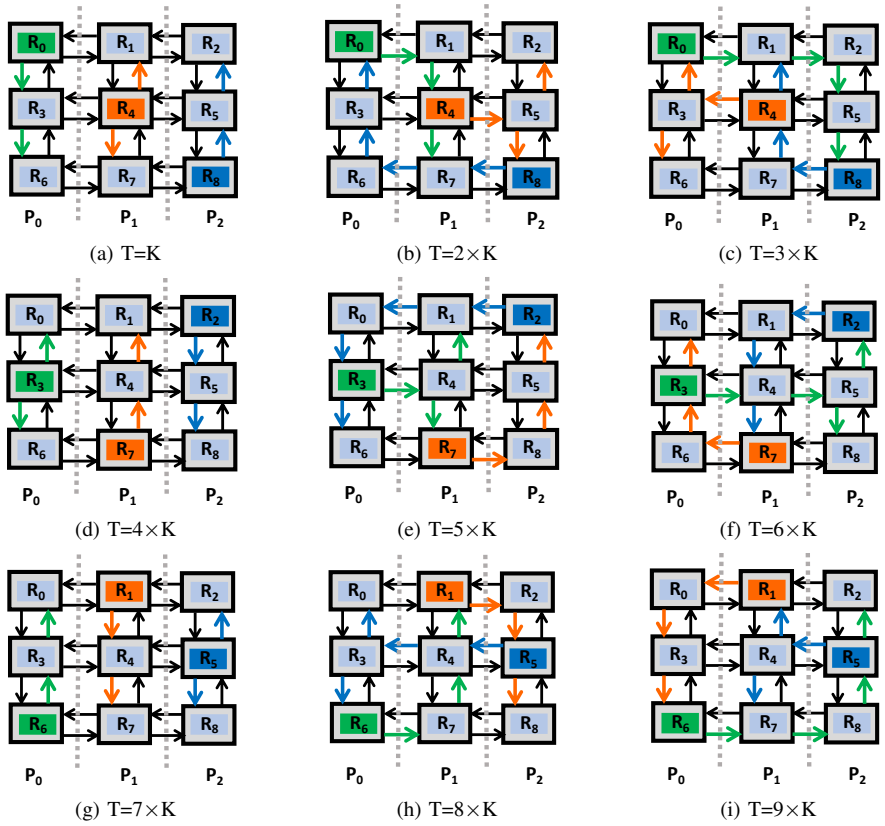


Figure 1: TDM-based non-overlapping paths: To ensure no collisions between packets, `FastPass` breaks the topology into P separate partitions (i.e., 0, 1 and 2) each of which has one prime router (i.e., green, orange and blue); First phase (a, b, c): (a) In the first K cycles, the green prime router can cover the routers located in P_0: R_0, R_3, and R_6; the orange one has access to the P_1 routers: R_1, R_4, and R_7; the blue prime router has an isolated path to the P_2 routers: R_2, R_5, and R_8. When the first time slot elapses, the links of each partition switches to handle the packets of other primes; (b) Links of P_0, P_1, and P_2 propagate the packets from blue, green and orange primes, respectively. (c) In this step, the links of P_0, P_1, and P_2 handle packets from orange, blue and green primes, respectively. this switching continues until each prime has covered all the routers in the network, indicating the completion of the phase. Second phase (d, e, f) and third phase (g, h, i) follow the same approach.

must send the signal N cycles ahead of the phase schedule for a N×N mesh to account for the N hops needed for the signal to reach the first row from the last row. This switching continues to give each router the opportunity to be the prime. Thus, the completion of each phase requires a fixed number of steps which depends on the number of partitions. Each step also needs a fixed number of cycles (K) to complete. This time slot is set at design time.[1]

2) `FastPass` *for throughput enhancement:* Each prime router has multiple chances to provide FastFlow traversal for its packets over the FastPass-Lanes. Each FastPass-Lane provides an isolated path from each prime router to the routers of one single partition. For each FastPass-Lane, each prime router examines each input buffer in a round-robin fashion (e.g., $injection$ buffers, $South$, $North$, $East$ and $West$) to see if there is a packet at the head of input buffer destined for a router located in the corresponding partition. Once such a packet is found for the input buffer,

[1]Qn 5 discusses how this value is pre-computed.

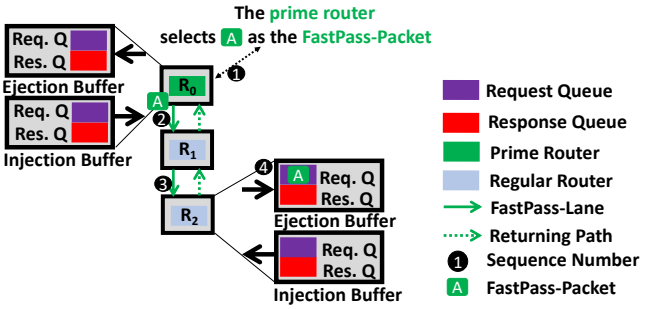


Figure 2: The prime router (i.e., R_0) upgrades a request packet (A) to a FastPass-Packet so that it can traverse the current FastPass-Lane to reach the intended destination (i.e., R_2). On the arrival, the packet can be ejected to the respective ejection queue.

the prime router upgrades it to the FastPass-Packet by sending it through FastFlow to reach the destination. Upon arrival at the destination, the FastPass-Packet is ejected by the network interface assuming the respective ejection queue

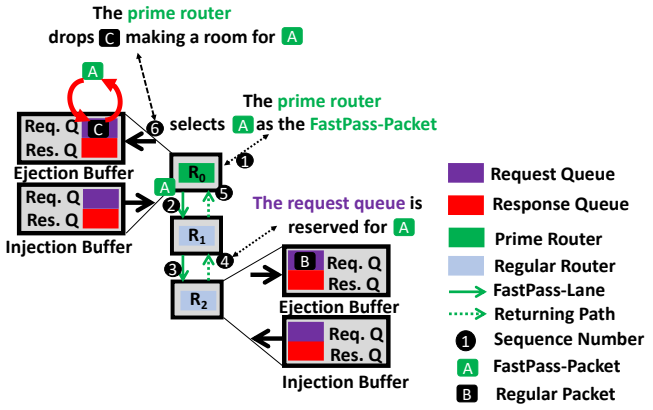


Figure 3: When the ejection queue of the destination router is full: The prime router (i.e., R_0) selects a request packet (A) as a FastPass-Packet. However, it faces a full ejection queue when it reaches the destination (i.e., R_2). In this step, the destination router reserves the intended queue for A. A goes back to the prime router to reside in the request injection queue of the prime. However, the queue is full with an injection packet (i.e., C). The prime router drops C to make a room for the rejected FastPass-Packet (A). Note that if the request injection queue is not full, no dropping is required.

is not full (Fig. 2).[2] Thereafter, the prime router examines the remaining input buffers in a round-robin manner for the current FastPass-Lane. Note that the time slot (K) to use of each FastPass-Lane is fixed and large enough to give the prime router sufficient time for checking all the input buffers and sending out the FastPass-Packets over the path.[3] In the next time slot, when the prime router has access to another FastPass-Lane, the router follows the same procedure. When each phase elapses, the prime status moves to the next set of pre-defined routers. Since the time slot (K) is fixed, all the new prime routers are enabled simultaneously. This procedure continues indefinitely through every router.

Qn1: Does a packet need to wait for the desired FastPass-Lane to be available? As mentioned in Sec. III-C1, each prime router eventually covers all routers in the network during each phase. If the current FastPass-Lane does not cover the intended destination for a packet, the packet needs either to wait until the intended FastPass-Lane becomes available or can traverse the router through the *regular pass*. Note that the regular traversal is always available unless there is a deadlock. In the case of deadlock, the intended FastPass-Lane eventually becomes available making forward progress for the deadlocked packet. *Note that in the absence of deadlocks during low load, packets do not wait for a FastPass-Lane to become available; FastPass kicks in as load increases.*

3) `FastPass` *as Deadlock Removal with No Virtual Networks:* Apart from the throughput enhancement, `FastPass` provides both protocol- and network-level deadlock freedom. Deadlock freedom is guaranteed because of the following: (1) eventually each router gets a chance to be upgraded to a prime router, (2) a prime router can upgrade the packet at the head of each input buffer to a FastPass-Packet, regardless of its message type, (3) when each time slot elapses, the prime router has access to a different FastPass-Lanes until it covers all routers in the network. As a result, the movement of all packets along the desired FastPass-Lane is guaranteed. Therefore, `FastPass` ensures that any packet of any message class will eventually have the opportunity to reach its destination router through the `FastPass`. Sec. III-D provides a detailed proof of correctness. In addition, `FastPass` allows multiple prime routers in the network–one per partition. Allowing multiple prime routers results in resolving multiple protocol-level deadlocks simultaneously.

4) Dynamic Bubble: Although the prior work has shown that the ejection queues are significantly underutilized and the ejected packets are consumed almost immediately [13], we need to consider what would happen if the ejection queue is full. When the FastPass-Packet arrives at the destination router, it is possible that the respective ejection queue is full, preventing the FastPass-Packet from being ejected. Since the main goal of this work is increasing throughput, a prime router increases the credit for the upstream router (i.e., indicating available slot) as soon as a FastPass-Packet departs the router. Therefore, it is not possible for a FastPass-Packet facing a full ejection queue (at the destination) to return to its prime router and reoccupy its previous buffer entry. To handle this challenge, we use a *subtle* dropping approach providing a *bubble* within the prime router. In this approach, some packets are potentially *droppable*.

To make the bubble, the prime router drops the *injection request packet* residing in the *request injection queue*, making an empty entry for the rejected FastPass-Packet (Fig. 3).[4] Note that if the *request injection queue* is not full, no dropping happens. Thus the FastPass-Packet is routed back to its prime router through the minimal returning path and then resides in the recently created, available slot. Note that collision is not possible between the FastPass-Lanes and the returning paths as shown in Fig. 4. Also the destination router pro-actively reserves the intended queue of the ejection buffer for the rejected FastPass-Packet once the intended ejection queue gets free space.[5] Not until the rejected FastPass-Packet resides in the intended ejection queue at the destination router, are other packets allowed to use it. Therefore, using the dropping and reservation approaches, the FastPass-Packet is eventually guaranteed to be accepted by the intended destination router. The proposed dropping approach has the following distinctive characteristics setting it apart from other methods [17], [33]:

[2]Sec. III-C4 and Qn 4 discuss when ejection queues is full.

[3]See Qn 5 for more details.

[4]If the rejected FastPass-Packet is a data packet, then multiple request injection packets might be dropped.

[5]Qn 4 discusses that the ejection queues can never be full indefinitely.

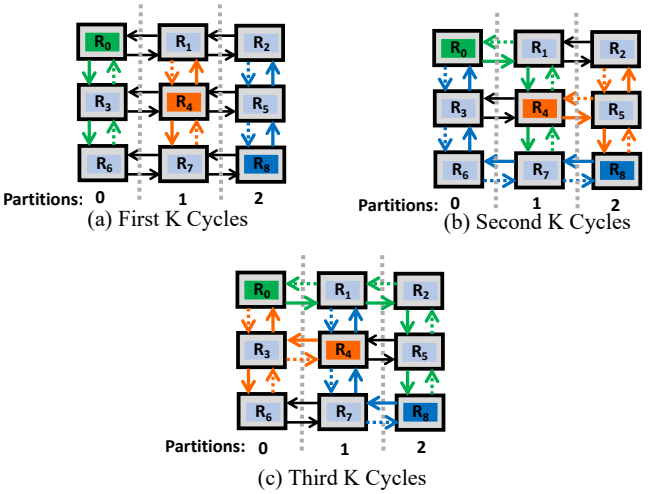


Figure 4: The FastPass-Lanes and the returning paths when the routers located on the diagonal are the prime routers. Dotted lines represent the returning paths for FastPass-Lanes. There is no overlap between these minimal paths.

1) The dropping happens when a FastPass-Packet encounters a full ejection queue at the destination router and a full request injection queue at the prime router. The likelihood of such a sequence of events is rare.
2) Since only the *injection request packet* is dropped, it can simply be regenerated using the information provided by the *miss status handling registers (MSHRs)* without changing the coherence protocol or using extra buffers. Note that the MSHRs keep track of what misses a core is waiting for. *Since the dropped packet's MSHR is local, the router does not have to communicate with a distant node to regenerate the packet.* Hence the dropped request packet can be easily regenerated.
3) Last but not least, the dropped request packet has not left the source router yet. Therefore, this dropping will not hurt performance.

Qn2: Is it possible for the rejected FastPass-Packet, residing in the request injection queue of the prime router, to be dropped to make a room for a new rejected packet? No: after that the rejected FastPass-Packet occupies the request injection queue of its prime router, it has two options to make forward progress: *traverse as a regular packet or traverse as a new FastPass-Packet.* For the latter, there are two possible scenarios:

1) The rejected packet was the last packet that the prime router upgraded; thus the router is not prime during the current phase. In this scenario, the packet needs to wait until the router turns into the prime again *if a regular traversal is impossible.* To ensure that we never drop a rejected FastPass-Packet, in a given FastPass-Lane, each prime router always starts examining the injection buffers first (starting with the request injection queue) and then moves on to the remaining input buffers.
2) The router is still prime and is going to examine other remaining input buffers for the *current FastPass-Lane*. In other words, the prime router has checked the injection buffer and it is the turn of the other input buffers. To avoid dropping the previous rejected packet, the previous rejected packet pro-actively moves to the empty slot which is available when the new FastPass-Packet departs the prime router. To provide such a transfer, the router microarchitecture includes connections between the injection buffer and all input buffers.[6]

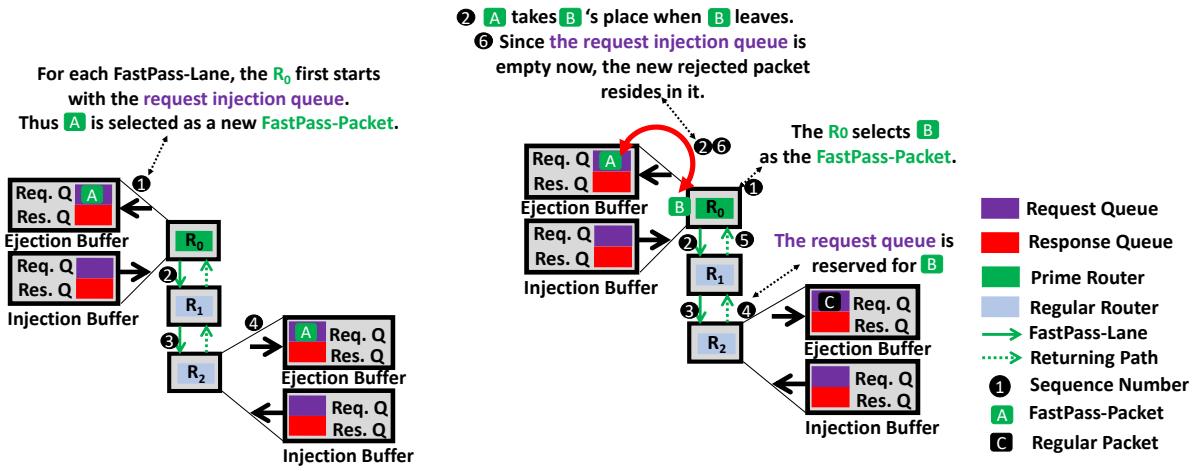
Therefore, `FastPass` guarantees dropping will only occur if needed for the *injection request packets*, which have not left the source router yet. Both of these scenarios are shown in Fig. 5.

Qn3: What if the ejection port is busy when the FastPass-Packet arrives? This situation happens when the FastPass-Packet arrives at the destination router while another packet is in the middle of ejection process. There are two scenarios for this case:

1) The first scenario is that the respective ejection queue of the destination router was reserved for the FastPass-Packet. This means that the FastPass-Packet faced a full ejection queue at the destination router causing the intended queue to be reserved for the packet. The packet reaches the destination again while another packet is being ejected to a different queue (not the reserved one). In this case, the FastPass-Packet is given higher priority over the ongoing ejection; the ongoing ejection is stalled and the FastPass-Packet is ejected to the respective ejection queue. Not until the FastPass-Packet gets ejected from the network can the current ejection packet use the ejection port. Since the intended ejection queue for the FastPass-Packet was reserved before, it is impossible that the stalled ejection packet is aiming for the same queue.
2) The second scenario happens when the respective ejection queue of the destination router was not reserved before. In other words, this is the first time that the FastPass-Packet arrives at the router. In this case, as discussed in Sec. III-C4, while the FastPass-Packet goes back to reside in the request injection queue of the prime router, the destination router reserves the intended ejection queue for the FastPass-Packet. Therefore, the FastPass-Packet is guaranteed to be ejected on its next arrival.

Qn4: Is it possible that the intended ejection queue never gets free space preventing the FastPass-Packet from getting ejected indefinitely? It is impossible for the ejection queues to be full indefinitely. As there are separate

[6]See Sec. III-E for more details.



(a) First scenario

(b) Second scenario

Figure 5: Under no circumstances is a rejected FastPass-Packet dropped: (a) For each FastPass-Lane, the prime router always starts with checking the request injection queue. Thus the previously rejected packet (e.g., A) is always selected as a new FastPass-Packet. (b) The prime router still needs to check other input buffers for the current FastPass-Lane. The prime router selects B as the FastPass-Packet. As soon as B departs the prime router, A takes the available space in the input buffer, making the request injection queue empty. This allows us to accommodate another rejected FastPass-Packet in the prime router.

ejection queues per message class, the ejection queues will never be full for an unlimited time. For each protocol transaction, there will always be at least one *sink* message class (e.g., response messages) that ends the transaction. Thus the ejection queues for sink message classes can always be consumed. Ejection queues of request messages may fill up if the processor awaits a response message deadlocked elsewhere in the NoC, stalling the processor. However, the response message will arrive eventually, unstalling the processor; receipt of the response message will allow the processor to restart consuming the request messages, freeing up the request buffer slots. Since `FastPass` allows multiple prime routers and FastPass-Packets in the network at any time, ejection queues are guaranteed to eventually have free space.

Qn5: What is the number of cycles between switching from one FastPass-Lane to another? As mentioned in Sec. III-C1, `FastPass` creates multiple non-overlapping minimal FastPaths each of which covers one partition for a fixed time slot (K). This time slot should be large enough so that a packet can make a round trip to the furthest destination in the network (network diameter). To define the fixed time slot, we use following metrics: the maximum number of hops in the topology, the number of input ports per router, and the number of virtual channels per input buffer. This pre-defined static time-bound is calculated as follows: $(2 \times \#Hops) \times \#Inputs \times \#VCs$. This time slot is sufficient for each prime router to check all the input buffers to enable FastFlow traversal for each FastPass-Lane. This value is set at the design time.

5) Lookahead: `FastPass` guarantees that the arrival time of FastPass-Packets at the destinations is fixed. This means that FastPass-Packets are always granted the output port while traversing the FastPass-Lane. The FastPass-Packet is given higher priority over regular packets by the routers. This approach is needed to have the FastPass-Packets make forward progress every cycle. To this end, the FastPass-Packets need to suppress the movement of regular packets through the intended output port at the next downstream router. To make this happen, `FastPass` employs a lookahead signal similar to prior work [19], [20]; when a prime router upgrades a packet to a FastPass-Packet, it enables the lookahead signal on the link to the downstream router one cycle ahead of the traversal. The signal informs the downstream router of the intended output port so that the downstream router can reserve the output port for the upcoming FastPass-Packet. Since `FastPass` uses minimal routing, the intended output port at the downstream router can be pre-computed. Therefore each router receiving the lookahead signal can update the signal and then forward it to the next downstream router.

The lookahead signal includes the destination ID and the output port ID. Assuming an 8×8 mesh, this information requires 10 bits. Since this information is also carried by the head flit, `FastPass` uses the first 10 bits of the datapath as lookahead.

D. Proof of Correctness

Lemma 1. *Every packet selected for FastFlow is guaranteed to reach its destination.*

Proof: During each phase, `FastPass` creates multiple non-overlapping FastPass-Lanes for a prime router which

covers all routers in the network. When a prime router upgrades a packet to a FastPass-Packet, it traverses the FastPass-Lane bufferlessly to reach its destination which lies on the current covered partition. Since the FastPass-Packet takes precedence over the regular packets, it is guaranteed that the FastPass-Packet makes forward progress every cycle with the help of lookahead signal. Therefore FastPass guarantees that the FastPass-Packet will arrive at the intended destination. *Note that getting ejected is not guaranteed yet.* ■

Lemma 2. *Every packet is eventually guaranteed to be selected for FastFlow traversal.*

Proof: During each phase, FastPass allows a set of prime routers in the network–one per partition. On the completion of each phase, the prime ability is given to the next adjacent router within the partition. This switching continues indefinitely so that all routers in the network will have a chance to be prime. As a result, since FastPass gives each router a chance to be a prime router, all the routers will be able to enable FastFlow for the packets of their input buffers. ■

Lemma 3. *There will eventually be free space in the ejection queues.*

Proof: Ejection queue will never be full indefinitely since: (1) there are separate queues in the ejection buffer–one per message class, (2) each protocol transaction has at least one sink message (e.g., response message class) ending the transaction; thus the ejection queues for sink message classes can always be consumed, (3) in the case that the ejection queue for request messages is full and the processor does not process requests since it awaits a response message, the response message will eventually arrive at the destination using FastFlow (Lemma 1 and Lemma 2); receipt of the response, will lead to consuming the requests, creating free space in the request ejection queue. ■

Lemma 4. *Since each FastPass-Packet will eventually get ejected, freedom from protocol- and network-level deadlocks is guaranteed.*

Proof: When a FastPass-Packet arrives at its destination, it will get ejected if the respective ejection queue has enough space. Otherwise it will get back to its prime router to reside in the request injection queue of the prime. Once the intended ejection queue of the destination router gets a free space (Lemma 3), the queue is pro-actively reserved for the packet. Thereafter, the packet will eventually be selected as a new FastPass-Packet to reach the destination again (Lemma 1 and Lemma 2). On arrival at the destination, it will get ejected to the reserved ejection queue successfully. Therefore, both protocol- and network-level deadlocks are guaranteed to be resolved. ■

Qn6:What if request packets consume all of the buffers in the network preventing responses from getting injected to the NoC? Since each prime router always starts examining the injection buffer and then moves on to the other input buffers to select the FastPass-Packets per FastPass-Lane, the response packet will be upgraded to a FastPass-Packet to reach its destination via FastFlow. Upon receiving the response packet, the processor will resume processing the requests, creating free spaces in the request ejection queue. Therefore FastPass can make forward progress for all the responses residing in the response ejection queue by upgrading them to the FastPass-Packets.

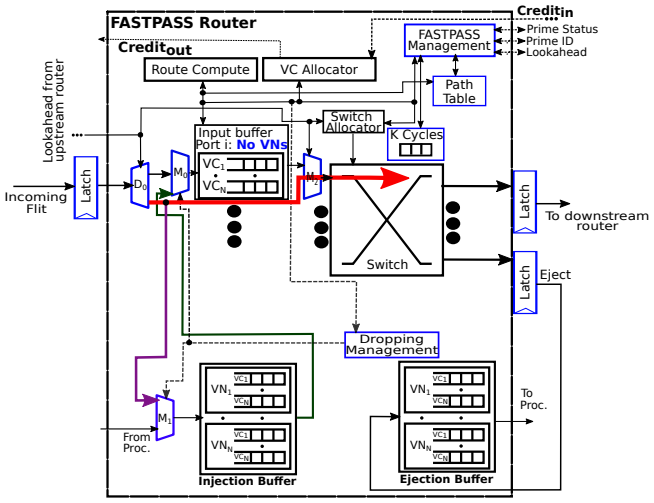


Figure 6: FastPass architecture; For simplicity, only one direction is represented. An incoming FastPass-Packet bypasses the input buffers (red path). An incoming rejected FastPass-Packet can reside in the request injection queue (purple path); the rejected FastPass-Packet can be transferred to the input buffer as soon as a new FastPass-Packet departs the router (green path).

E. Router Microarchitecture

Fig. 6 depicts the FastPass router microarchitecture and its network interface controller (consisting of the injection and ejection buffers).

Minimal FastPass-Lanes and returning paths: The FastPass-Lane table has P entries where P is the number of partitions. For an 8×8 mesh, it translates into 3-bits for each entry. We use a pointer which determines the partition (i.e., column) that the prime router can have access to via the FastPass-Lane. If a prime router has a packet whose destination is located in the current partition,[7] it promotes the packet and then sends it out via the XY routing. In the next time slot, the pointer determines the next partition.

To route a rejected packet back to its prime router, the destination router uses the $PrimeID$[8] (6 bits for an 8×8 mesh) to find the column where the prime router is located

[7]Route compute unit provides this information.

[8]The $PrimeID$ is sent a cycle ahead similar to the lookahead signal.

in. Then the rejected packet is routed back to the prime using YX routing.

To guarantee no collision between the FastPass-Lanes and the returning paths, the concurrent prime routers should not be in a same row and in a same column. All other arrangements with above minimal routing algorithm lead to non-overlapping paths with no collision.

Upgrading routers: `FastPass` management is responsible for upgrading a router to prime when it is its turn. For each prime router, there are multiple non-overlapping FastPass-Lanes over which the FastPass-Packets traverse to reach the destinations located on the currently covered partition. After each K cycles, the current partition is changed to another one, leading to a different FastPass-Lane. When the prime routers are done with the input buffers for all partitions (all routers in network), the next set of routers are selected as new primes in a statically determined order. To this end, the `FastPass` management unit uses a side-band signal (i.e., one bit $PrimeStatus$). To enable the next set of prime routers at a same time, recall that a prime router in the last row must send the signal N cycles ahead of the phase schedule for a N×N mesh to account for the N hops needed for the signal to reach the corresponding router in the first row.

Upgrading packets: `FastPass` management of a prime router also upgrades a regular packet of its input buffers to a FastPass-Packet using the M_2 multiplexer if its destination is located on the current partition. Note that M_2 is responsible to direct the incoming FastPass-Packet from the upstream router, the upgraded packet for the current router, and the regular packets of the input buffers.

Lookahead: After upgrading the packet, one cycle ahead of the FastPass-Packet traversal, the `FastPass` management sends the lookahead signal to the downstream router. The lookahead signal is needed to set the multiplexers and demultiplexers of the downstream router so that the FastPass-Packets take precedence over regular ones. Thus the output ports are always granted to the FastPass-Packets. The incoming FastPass-Packets bypass the router through D_0 and M_2. As mentioned earlier, `FastPass` uses the first 10 bits of the datapath as lookahead for an 8×8 mesh.

Making bubble in a prime router: When a rejected FastPass-Packet is heading to its prime router using the returning path, it is sent to the request injection queue of the prime via M_1 (purple path). If the request queue is full, the dropping management drops the request packet making room for the rejected packet. Note that if another packet is getting buffered through the injection port, the rejected packet takes precedence over that. After the rejected packet leaves the router, the dropped packet is re-injected with the help of MSHRs.

Freeing up the request injection queue from the rejected FastPass-Packet: The rejected packet can be transferred to the input buffer (the green path) as soon as a new FastPass-Packet departs the prime. Note that the rejected packet can be selected as a new FastPass-Packet whether residing in the injection queue or the input buffer. It also can traverse the router through the regular pass while residing in either buffer.

Virtual networks: As shown in Fig. 6, `FastPass` does not use any VNs for the input buffers. Protocol-level deadlock freedom is guaranteed with no VNs. However, it still has one queue per message class in the injection and ejection buffers similar to the prior work [13], [24].

Table II: Key Simulation Parameters.

Core	16 and 64 cores, x86 ISA, 1GHz, OoO, 8-Wide, ROB size:192
L1 Cache	private, 32KB Ins. + 64KB Data, 2-way set assoc.
LLC	shared, distributed, 2MB, 8-way set assoc.
Cache Block Size	64B
Cache Coherence	MOESI Hammer
Topology	4×4, 8×8, and 16×16
Evaluated Schemes	EscapeVC [8], SWAP [26] (swap duty: 1K cycle) SPIN [31] (detection threshold: 128 cycles) DRAIN [24] (DRAIN period: 64K cycles) Pitstop [13], MinBD [12], TFC [19]
Routing Algorithm	SWAP, SPIN, DRAIN, Pitstop, `FastPass` (fully adaptive routing) EscapeVC: west-first within escape VC and fully adaptive within other VCs, TFC: West-first minDB: Deflection routing
Router Latency	1-cycle
Number of VNs	0-VN (`FastPass` and Pitstop) and 6-VN (EscapeVC, SPIN, SWAP, DRAIN, TFC)
Number of VCs	`FastPass` (1, 2, 4) per input buffer EscapeVC, SPIN, SWAP, DRAIN, MinBD, TFC, Pitstop (2 VCs)
Buffer Size	5-flit
Link Bandwidth	128 bits/cycle
Flow Control	VCT – Single packet per VC
Synthetic traffic	Uniform, Transpose, and Shuffle – Mix of 1-flit and 5-flit

F. `FastPass` *in Irregular Topologies*

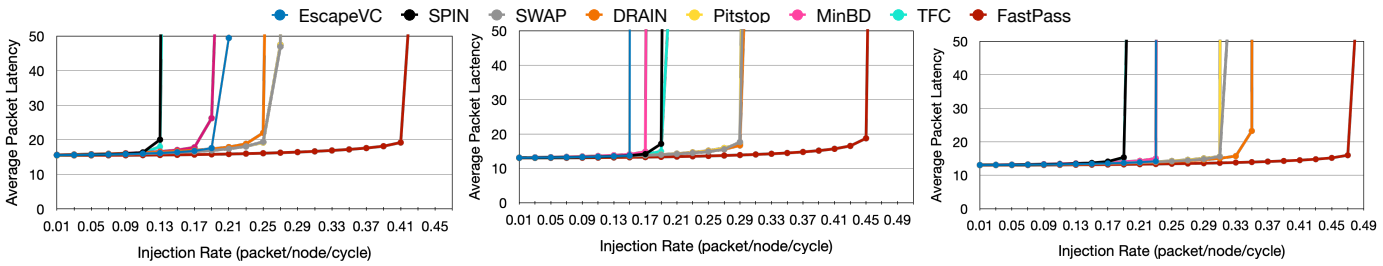
`FastPass` is agnostic to the topology types. To handle irregular topologies, we can leverage algorithms from prior work [24] that can find holistic paths that are guaranteed to traverse every physical link in the network exactly once. Such algorithms are applicable to any arbitrary topology as long as all channels between routers are bidirectional (i.e., two opposing unidirectional physical links). Segmenting a holistic path is guaranteed to produce a set of non-overlapping paths, which `FastPass` can use to derive its partitions.

IV. Evaluation

We evaluate `FastPass` using full-system *gem5* [4] with the *Garnet2.0* [1] network model and the *Ruby* memory model to run PARSEC [3] and SPLASH-2 [37] applications. Table II lists the key configuration parameters used.

A. *Synthetic traffic*

Fig. 7 draws a comparison between `FastPass` and the baselines for different synthetic traffic patterns when using 4 VCs per input buffer across an 8×8 mesh. As shown, `FastPass` outperforms all the schemes under comparison; the main reason for this improvement is that `FastPass`



(a) Transpose (b) Shuffle (c) Bit Rotation

Figure 7: Performance of the schemes under comparison and FastPass for synthetic traffic.

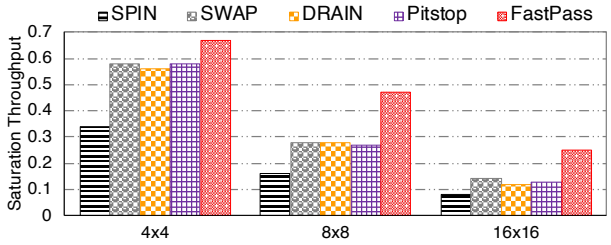


Figure 8: Saturation throughput as the network size increases.

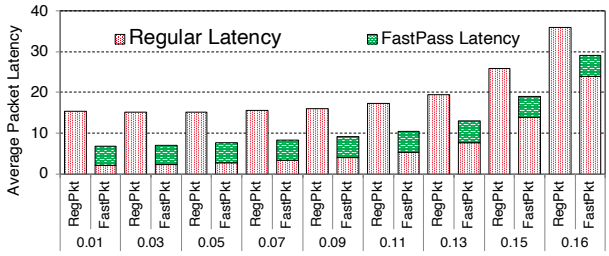


Figure 9: Breakdown of the latency distribution for FastPass-Packets and regular packets across uniform traffic using 1 VC.

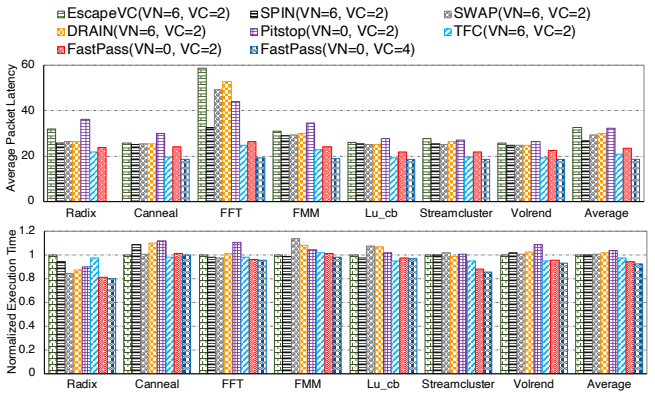


Figure 10: Average packet latency and execution time (normalized to EscapeVC).

provides multiple FastPass-Lanes in the network over which FastPass-packets can traverse bufferlessly, providing higher saturation point compared to other techniques. On average, FastPass provides up to 51% higher throughput over periodic deadlock solutions (SWAP, Pitstop, and DRAIN), 1.8× over SPIN and TFC, and 1.4× over MinBD.

Throughput across different topology sizes: Fig. 8 demonstrates saturation throughput for *Transpose* traffic using 4-VCs as the network size increases. As shown, FastPass outperforms the baselines in all the network sizes. Since the number of prime routers determines the number of FastPass-Packets at a time, the throughput enhancement in FastPass increases as the network size does. SPIN has the lowest throughput in all the cases because its deadlock detection imposes considerable latency overhead at saturation to detect deadlock cycles. In summary, compared to SWAP, FastPass provides 17% higher throughput in 4×4, 67% in 8×8 and 78% in 16×16 networks.

Latency of regular packets vs. FastPass-Packets: Fig. 9 shows the breakdown of the average packet latency for FastPass-Packets and regular packets for different injection rates. The latency of the FastPass-Packets consists of two components: regular time (i.e., buffered time) and FastPass time (i.e., bufferless time). To collect the result, we use *Uniform* traffic with 1 VC. At low injection rates, the fraction of the regular time for FastPass-Packets is small. As the load increases, the regular time makes up the majority of the latency for the FastPass-Packets. As shown, the bufferless time of the FastPass-Packets remains small across all the injection rates (including post saturation).

B. Application Traffic

Fig. 10 shows average packet latency and execution time of FastPass and the baselines. Pitstop and FastPass use no VNs while other schemes use 6-VNs to avoid protocol-level deadlocks in MOESI Hammer protocol. Note that DRAIN can function correctly with no VNs; however, it requires a large amount of buffers which is not minimal [13]. On average, FastPass with 2-VCs provides 6% improvement on execution time over network-level deadlock solutions (EscapeVC, SPIN, and SWAP), 8-9.5% over protocol-level deadlock solutions (DRAIN and Pitstop), and 3% over TFC. FastPass with 4 VCs improves execution time by 5% over TFC at 1/3th buffer area/power budget. In terms of average packet latency, FastPass provides up to 46%

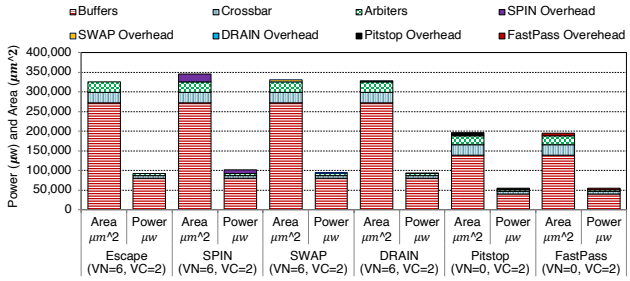


Figure 11: Post Place-and-Route router power and area (28nm TSMC, 1GHz).

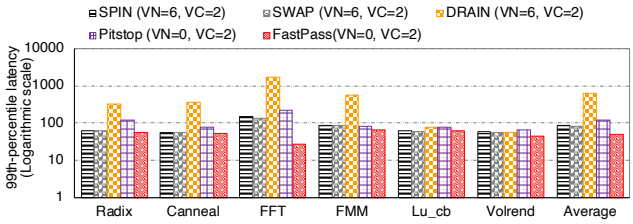


Figure 12: 99^{th} percentile tail latency (Logarithmic scale).

improvement over prior art.

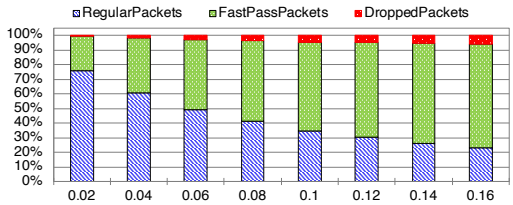
C. Area and Power

As mentioned in IV-B, FastPass and Pitstop use no VNs but other methods need to use 6 VNs due to providing correctness or minimizing the amount of buffer space. Fig. 11 shows the breakdown of static power and area used by FastPass and the baselines using TSMC 28nm. FastPass reduces power and area by 41% and 40% over EscapeVC. FastPass has similar area and power consumption as Pitstop. SPIN imposes 6% area overhead compared to EscapeVC due to its deadlock detection circuit. The area overhead of FastPass consumes only ∼4% of FastPass area.

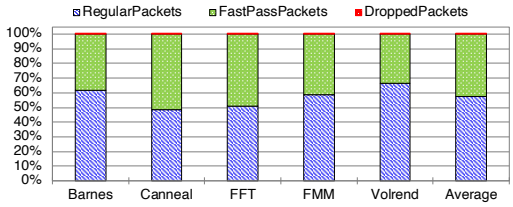
D. Sensitivity study

Tail latency: Fig. 12 shows the worst-case latency (i.e., 99^{th} percentile tail latency) of packets in the network. FastPass with 2 VCs has the lowest tail latency compared to all baselines. This is because FastPass allows multiple FastPass-Packets at any given time bypassing congested areas. DRAIN has the worst tail latency due to its misrouting; misrouted packets may create deadlocks which must wait until next DRAIN period to arrive.

Fraction of dropped packets: Fig. 13 shows the breakdown of different packet types in FastPass: FastPass-Packet, regular, and dropped packets. Fig. 13(a) shows *Uniform* traffic using 1 VC. During low loads, regular packets are dominant–FastPass operates mostly like baseline. As the load increases, FastFlow kicks in, increasing the number of FastPass-Packets. Note that the fraction of dropped packets



(a) Uniform



(b) Real Workload

Figure 13: Breakdown of the different packet types using 1 VC under (a) uniform traffic and (b) application traffic.

is negligible even at post saturation (5.9%). In real applications, the fraction of dropped packets is 0.3%. As a comparison, SCARAB [17] drops up to 9% of the total packets *mid-transmission* for real applications.

V. Related Work

A. Performance Optimizations

Credit flow control: This flow control employs credits so that the upstream router can track of the number of available buffer slots at the downstream router. Only when there are available slots can the upstream router forward a packet to the downstream router. Prior work improves on this by enabling a bufferless bypass approach. This type of approach eliminates the need for arbitration and buffering for the packet traversal. Token Flow Control (TFC) [19] employs tokens indicating resource availability in the network. Each router broadcasts a token along a fixed length so that packets can bypass the congestion. However, in the case of conflict between two packets, one packet must get buffered. In Express Virtual Channels (EVCs) [20], packets can skip the entire router pipeline by creating straight bypass paths. Packets cannot bypass routers when changing dimensions. Both TFC and EVCs provide opportunistic bypass paths where every packet is not guaranteed to use the bypass approach.

SDM- and TDM-based flow control: Circuit-switched coherence [11] partitions each link into multiple planes in a space-division multiplexing (SDM) approach. Packets can opportunistically travel bufferlessly in a circuit-switched manner in an assigned partition; however, in the event of contention, the packet must be buffered and cannot move to an idle partition. Prior work improves on this by enabling TDM [38]. During each time slot, the links are configured to handle either packet-switched or circuit-switched connections.

Deflection flow control: This type of solution [12] relies on misrouting packets; in these techniques if multiple flits contend for the same output port, one of them is given the requested output port and others are deflected (misrouted). Due to the misrouting, they suffer from throughput degradation.

Drop-based flow control: Instead of misrouting packets, BPS [33] proposes a dropping approach when multiple packets contend for the same output port. In this case, one packet is given the intended output port while the other one is dropped. As a result, this method suffers from frequent dropping. SCARAB [17] reduces the number of dropped packets by using MSHRs as a temporary storage and imposing a dedicated circuit-switched network to enable packet re-transmission. Both BPS and SCARAB need to have an additional buffer per router for possible re-transmission.

B. Protocol-level Deadlock Freedom Solutions

A certain number of VNs are required to prevent protocol-level deadlock. The minimum number of required VNs is determined by the cache coherence protocol. The larger the number of message classes, the larger the number of VNs required. Since VNs are implemented using memory elements, a large number of VNs leads to considerable power and area overhead. Most of the prior work [2], [26], [31], [32] uses costly VNs as the primary de facto solution to avoid protocol-level deadlocks. Therefore, there is an opportunity to reduce this cost without sacrificing correctness and performance.

Some prior work maintains the protocol-level deadlock freedom with no VNs. mDisha [34] adds extra buffers in the network to detect protocol-level deadlock and then resolve it. Bubble Coloring (BC) [35] statically reserves two packet-sized bubbles per router in a virtual ring. This approach results in frequent misrouting. DRAIN [24] periodically misroutes packets in the network to break potential deadlock cycles. Pitstop [13] fundamentally improves on this by providing a bypass mechanism between network interfaces to avoid misrouting. However, only one message type can use the bypass approach in the network at a time, leading to performance issues as the network size increases. SEEC [27] provides simultaneous bufferless paths like `FastPass`. However, `FastPass` is free from sending tokens (i.e., seekers) and its associated overhead to upgrade packets.

C. Network-level Deadlock Freedom Solutions

Applying turn restrictions to the routing algorithm avoids network-level deadlock at the cost of limiting path diversity. EscapeVC [8] improves on this by imposing an additional VC (per VN) as the escape virtual channel. However, packets within the escape VC cannot exploit full path diversity. Other solutions eliminate the need for an extra VC by using deadlock detection and recovery mechanisms [2], [31]. This type of solution needs global coordination and synchronization between routers leading to significant overhead. Another type of solution eliminates the need for detecting deadlock by employing a periodic approach [25], [26], [28]. For example, SWAP [26] periodically makes forward progress for a blocked packet at the cost of misrouting. Bubble flow controls [6], [30] insert empty buffer slots (called bubbles) to prevent network-level deadlocks. None of the aforementioned methods can handle protocol-level deadlocks without using VNs.

D. `FastPass` *vs. Prior work*

In this section, we highlight the differences between `FastPass` and prior work.

Comparison against other performance optimization techniques: Unlike the flow control techniques such as EVCs and TFC [19], [20], `FastPass` guarantees that every packet can be upgraded to a FastPass-Packet to traverse bufferlessly over FastPass-Lanes. `FastPass` is also capable of using one single VC while TCF and EVCs need multiple VCs to provide bypass paths. `FastPass` also can provide both protocol- and network-level deadlocks freedom with no VNs while TCF and EVCs are only limited to deadlock-free algorithms.

Comparison against other TDM techniques: The TDM used in `FastPass` has differences with other TDM-based NoCs [29], [38]. For example, Yin et al. [38] require multiple setup phases to reserve and then remove a circuit-switched path, congesting the network. Unlike these methods, `FastPass` provides multiple non-overlapping predefined paths without global coordination making it free from such challenges. These methods also rely on multiple VCs [5], [29], [38] while `FastPass` can work with a single VC. *None of the these methods can resolve network- and protocol-level deadlocks.*

Comparison against other deadlock freedom techniques: Unlike the prior work which targets only deadlock freedom, `FastPass` guarantees both types of deadlock freedom and also provides a large throughput enhancement. `FastPass` has following unique characteristics setting it apart from prior work: (1) no VNs, (2) high performance, (3) no misrouting, (4) no scalability issue, (5) no deadlock detection, and (6) no additional buffers.

Comparison against highly-connected topologies: Highly-connected topologies [16], [18] provide performance gains through express paths at the cost of the physical variety. Furthermore, none of these topologies alone can guarantee protocol- and network-level deadlocks. Therefore, `FastPass` provides significantly better power and area results over those topologies.

VI. Conclusion

`FastPass` proposes a unified solution for head of line

blocking and deadlock problems. `FastPass` upgrades multiple packets to reach their destinations bufferlessly. To avoid any collisions between the upgraded packets, we provide non-overlapping pre-defined paths over which packets can make forward progress every cycle. Therefore, these promoted packets cannot be blocked by congestion nor deadlock. Unlike the prior work, `FastPass` does not use VNs, additional buffer/VC, deadlock detection, or misrouting.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and members of the NEJ Group for their helpful comments to improve this work. We thank the SPIN authors for sharing their gem5 implementation. Special thanks to Mayank Parasar for sharing the gem5 implementation of SWAP, MinBD, and TFC. The authors also would like to thank William Won for his help on placed-and-routed RTL implementations. This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Canadian Foundation for Innovation.

REFERENCES

[1] N. Agarwal, T. Krishna, L. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009.

[2] K. V. Anjan and T. M. Pinkston, "An efficient, fully adaptive deadlock recovery scheme: DISHA," in *Proceedings 22nd Annual International Symposium on Computer Architecture*, 1995.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, 2011.

[5] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip," in *Design, Automation and Test in Europe (DATE)*, 2005.

[6] L. Chen, R. Wang, and T. M. Pinkston, "Critical bubble scheme: An efficient implementation of globally aware network flow control," in *2011 IEEE International Parallel Distributed Processing Symposium*, 2011, pp. 592–603.

[7] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.

[8] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, Dec 1993.

[9] J. Duato and T. M. Pinkston, "A general theory for deadlock-free adaptive routing using a mixed set of resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pp. 1219–1235, Dec 2001.

[10] J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 10, pp. 1055–1067, 1995.

[11] N. Enright Jerger, L.-S. Peh, and M. Lipasti, "Circuit-switched coherence," in *International Symposium on Networks-on-Chip*, 2008.

[12] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu, "MinBD: Minimally-buffered deflection routing for energy-efficient interconnect," in *2012 Sixth IEEE/ACM NOCS*, 2012, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/NOCS.2012.8

[13] H. Farrokhbakht, H. Kao, K. Hasan, P. Gratz, T. Krishna, J. San Miguel, and N. Enright Jerger, "Pitstop: Enabling a virtual network free network on chip," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2021.

[14] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proceedings the 19th Annual International Symposium on Computer Architecture*, 1992.

[15] P. Gratz, B. Grot, and S. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2008.

[16] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *International Symposium on High Performance Computer Architecture*, 2009, pp. 163–174.

[17] M. Hayenga, N. Enright Jerger, and M. Lipasti, "SCARAB: A single cycle adaptive routing and bufferless network," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 244–254.

[18] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *International Symposium on Microarchitecture (MICRO)*, 2007, pp. 172–182.

[19] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2008, pp. 342–353.

[20] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: towards the ideal interconnection fabric," in *ISCA*, 2007.

[21] Z. Li, J. San Miguel, and N. Enright Jerger, "The runahead network-on-chip," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[22] S. Ma, N. Enright Jerger, and Z. Wang, "DBAR: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[23] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 network architecture," in *HOT 9 Interconnects. Symposium on High Performance Interconnects*, 2001.

[24] M. Parasar, H. Farrokhbakht, N. Enright Jerger, P. Gratz, T. Krishna, and J. San Miguel, "DRAIN: Deadlock removal for arbitrary irregular networks," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[25] M. Parasar, A. Sinha, and T. Krishna, "Brownian bubble router: Enabling deadlock freedom via guaranteed forward progress," in *2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2018, pp. 1–8.

[26] M. Parasar, N. Enright Jerger, P. V. Gratz, J. San Miguel, and T. Krishna, "SWAP: Synchronized weaving of adjacent packets for network deadlock resolution," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[27] M. Parasar, N. Enright Jerger, P. V. Gratz, J. San Miguel, and T. Krishna, "SEEC: Stochastic escape express channel," in *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2021.

[28] M. Parasar and T. Krishna, "Bindu: Deadlock-freedom with one bubble in the network," in *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, 2019.

[29] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation," in *Design, Automation Test in Europe(DATE)*, 2015.

[30] V. Puente, C. Izu, R. Beivide, J. Gregorio, F. Vallejo, and J. Prellezo, "The adaptive bubble router," *J. Parallel Distrib. Comput.*, vol. 61, no. 9, pp. 1180–1208, 2001.

[31] A. Ramrakhyani, P. V. Gratz, and T. Krishna, "Synchronized progress in interconnection networks (SPIN): A new theory for deadlock freedom," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*, 2018.

[32] A. Ramrakhyani and T. Krishna, "Static bubble: A framework for deadlock-free irregular on-chip topologies," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[33] C. G. Requena, M. E. G. Requena, P. J. L. Rodríguez, and J. D. Marín, "An efficient switching technique for NoCs with reduced buffer requirements," in *International Conference on Parallel and Distributed Systems*, 2008.

[34] Y. Song and T. Pinkston, "A progressive approach to handling message-dependent deadlock in parallel computer systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, pp. 259–275, 2003.

[35] R. Wang, L. Chen, and T. M. Pinkston, "Bubble coloring: Avoiding routing- and protocol-induced deadlocks with minimal virtual channel requirement," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, 2013, pp. 193–202.

[36] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip," in *International Symposium on Computer Architecture (ISCA)*, 2013.

[37] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995.

[38] J. Yin, P. Zhou, S. S. Sapatnekar, and A. Zhai, "Energy-efficient time-division multiplexed hybrid-switched NoC for heterogeneous multicore systems," in *International Parallel & Distributed Processing Symposium (IPDPS)*, 2014.