2021 27th International Symposium on High-Performance Computer Architecture

Pitstop: Enabling a Virtual Network Free Network-on-Chip

Hossein Farrokhbakht[1], Henry Kao[1], Kamran Hasan[1], Paul V. Gratz[2], Tushar Krishna[3], Joshua San Miguel[4], Natalie Enright Jerger[1]
[1]University of Toronto, [2]Texas A & M, [3]Georgia Institute of Technology, [4]University of Wisconsin-Madison
Email: [1]h.farrokhbakht@mail.utoronto.ca, [1]h.kao@mail.utoronto.ca, [1]kamran.hasan@mail.utoronto.ca, [2]pgratz@tamu.edu, [3]tushar@ece.gatech.edu, [4]jsanmiguel@wisc.edu, [1]enright@ece.utoronto.ca

***Abstract*—Maintaining correctness is of paramount importance in the design of a computer system. Within a multiprocessor interconnection network, correctness is guaranteed by having deadlock-free communication at both the protocol and network levels. Modern network-on-chip (NoC) designs use multiple virtual networks to maintain protocol-level deadlock freedom, at the expense of high power and area overheads. Other techniques involve complex detection and recovery mechanisms, or use misrouting which incurs additional packet latency. Considering that the probability of deadlocks occurring is low, the additional resources needed to avoid/resolve deadlocks should also be low. To this end, we propose Pitstop, a low-cost technique that guarantees correctness by resolving both protocol and network-level deadlocks without the use of virtual networks, complex hardware, or misrouting. Pitstop transfers blocked packets to the network interface (NI) creating a bubble (empty buffer slot) which breaks deadlock. The blocked packet can make forward progress through NI to NI traversals using low complexity bypassing mechanisms. This scheme performs better due to higher utilization of virtual channels and works on arbitrary irregular topologies without any virtual networks. Compared to state-of-the-art solutions, Pitstop can improve performance up to 11% and reduce power and area up to 41% and 40%.**

I. Introduction

Networks-on-Chip (NoCs) improve scalability over traditional bus-based or crossbar interconnects at the expense of increased complexity in maintaining correctness. A NoC design guarantees correctness by being free of deadlock–a cyclic resource dependence that can occur within the network or in the communication protocols between network nodes. Maintaining deadlock freedom not only adds considerable hardware and energy overheads in the form of virtual channels, but also incurs performance penalties due to turn restrictions or misrouting. In addition, the difficulty of maintaining correctness increases with shrinking technology size as smaller technology nodes accelerate link failures [32].

Deadlocks can be categorized into protocol- and network-level as seen in Figs. 1(a) and 2(a). In both cases, packets do not make forward progress, causing the system to stall indefinitely. Network-level deadlocks involve a cyclic dependency of network resources, whereas protocol-level deadlocks involve packets of different message types blocking one another.

Solutions to handle network-level deadlocks apply turn restrictions on all virtual channels (VCs) (e.g., XY and West-first routing) [15] or use escape VCs (i.e., partially adaptive routing shown in Fig. 2(b)) [18]–[20], [34]. These techniques cannot support irregular topologies (due to faulty links), and use costly virtual networks (VNs) and VCs to avoid deadlock. Other solutions rely on detection and recovery mechanisms [3], [55], [56]. For example, SPIN [55] detects potential deadlocks, then spins the packets to recover from deadlock. Such techniques introduce hardware overheads which limits scalability (Fig. 2(c)). Instead of detecting, bubble flow control techniques insert empty buffer slots to prevent deadlocks [11], [14], [53], [56], [64]. SWAP [48] fundamentally improves on this by recognizing that deadlocked packets can be exchanged in-place without needing extra buffers. Instead, SWAP periodically misroutes one packet to make forward progress on another (Fig. 2(d)). DRAIN [46] periodically drains all packets in the network to break potential deadlocks. Since draining may misroute packets, latency may increase, and new deadlocks may form and drive up the worst-case latency. Like SWAP and DRAIN, other prior work uses misrouting to solve deadlocks [24], [37], [43], [64]; however, they can incur high packet latencies. An ideal deadlock freedom scheme should be area and power efficient through low-cost hardware modifications, and also be performant by avoiding misrouting.

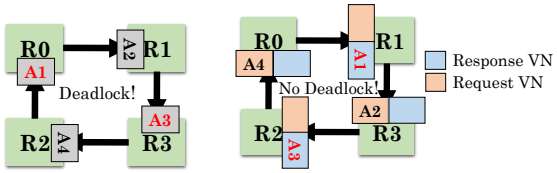


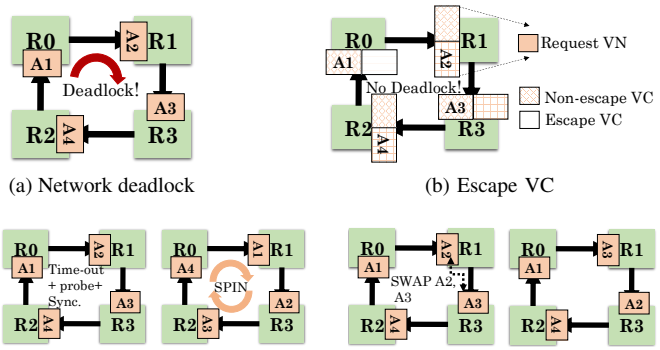
Fig. 1: (a) Protocol-level deadlock; A_1 and A_3 responses are blocked by A_2 and A_4 requests and vice versa leading to a cycle, (b) Current protocol-level deadlock solution: Assign one VN to each message class (i.e., request and response message classes for this figure).

VNs also prevent protocol-level deadlocks [44] by allocating different message types to different VNs to avoid one type blocking another (Fig. 1(b)). A VN is typically composed of multiple VCs to mitigate *head-of-line blocking*. As a result, they consume most of the NoC's power and area [29]. For example, the *MOESI Hammer protocol* requires six VNs for each input buffer just to guarantee protocol-level deadlock freedom; if it employs two VCs per VN, this leads to 12 VCs in total for each input buffer–imposing significant power and

TABLE I: Comparison of different deadlock freedom solutions.

Proposed Solutions	No Detection	**Protocol Deadlock Freedom**	Network Deadlock Freedom	Full Path Diversity	**Min. Buffer Space/Port**	**Low-power**	**Supports Wormhole**	**No Misrouting**
Turn Restrictions [15]	✓	✗	✓	✗	$\#VNs$	✗	✓	✓
Escape VCs [18]–[20], [34]	✓	✗	✓	✗/✓ **	$(\#VNs + 1)$	✗	✓	✓
Virtual Networks [44]	✓	✓	✗	✗	$\#VNs$	✗	✓	✓
SPIN [55]	✗	✗	✓	✓	$\#VNs$	✗	✗/✓ *	✓
SWAP [48]	✓	✗	✓	✓	$\#VNs$	✗	✗/✓ *	✗
DRAIN [46]	✓	✓	✓	✓	$\#VNs^{***}$	✗***	✗/✓ *	✗
Our Method: Pitstop	✓	✓	✓	✓	1	✓	✓	✓

* Cannot support Wormhole flow control without adding packet truncation support. ** Limited path diversity within escape VC.
*** Although DRAIN can work with no VNs, it needs a large amount of buffer space which is non-minimal (Sec. III-C).



(a) Network deadlock (b) Escape VC

(c) SPIN (d) SWAP

Fig. 2: Network-level deadlock and its exiting solutions: (a) Network-level deadlock; all packets (request type in this figure) hold buffers while they are waiting for each other, (b) Escape VCs: turn restrictions are applied only to the escape VC. Packets in the non-escape VCs take advantage of full path diversity, (c) SPIN: after SPIN's counter triggers a time-out, it sends probe packets to detect the deadlock path; then, it sends move packets to synchronize the routers for the spin. After that, all the routers send the packets at the same time, and (d) SWAP: SWAP breaks the deadlock at the cost of misrouting A_3.

area penalties. The number of required VNs is expected to increase as protocol complexity increases to handle heterogeneous devices in SoCs [66]. Since the dependency chain of the protocol messages is not only limited to the network, but also involves the directories and caches, detecting protocol-level deadlock is a challenging task.

None of the aforementioned techniques solve protocol-level deadlocks without the use of multiple VNs, deadlock detection, and/or misrouting. However, the probability of protocol-level deadlock occurring is low, even in the presence of faulty links. We observe lower utilization of network interface (NI) resources, specifically the ejection queue, compared to router input buffers making it a good candidate for temporary packet storage (Sec. III-D). We leverage these insights to propose a low-cost, VN-free scheme called *Pitstop* that guarantees deadlock freedom. Pitstop selects a blocked packet in a router and transfers it to the ejection queue. The blocked packet then makes forward progress through a bypass mechanism which also frees up router resources to break the deadlock. Pitstop is the first technique that eliminates both network- and protocol-level deadlocks without any virtual networks, extra buffers, detection and recovery mechanisms, nor misrouting. Tab. I summarizes the key differences between Pitstop and existing solutions. Compared to the state-of-the-art, Pitstop reduces energy up to 40% and improves performance up to 11%.

II. Background

Protocol-Level Deadlock: Coherence protocols have different classes of messages, and each type is associated with a set of coherence actions. Typically these message classes have priority in the order in which they must be processed otherwise deadlocks can occur. For example, in a simple request-reply system, replies should not be able to be blocked by requests, otherwise unreplied transactions can accumulate without being processed until the system runs out of resources – causing a protocol-level deadlock (Fig. 1(a)).

NoC Router: A generic NoC router is comprised of buffers, switch, arbitration, routing unit, and network interface (NI). In addition to the buffers in inter-router ports (e.g. $South$, $North$, $East$, and $West$), the NI consists of one *injection queue*, storing incoming messages from the processor, and one *ejection queue*, storing incoming messages from the inter-router input buffers [8], [21]. To avoid protocol-level deadlock, each message class (e.g., request, response) must traverse its own network [61]. Thus, all inter-router input buffers (e.g. $South$ buffer) and NI buffers (i.e., injection and ejection queues) must use different VNs per message class [21], [61].

Network-Level Deadlock: A packet from a router's input buffer can only be sent if the downstream buffer has space. Situations may occur where the forward progress of one packet depends on the downstream packet which then depends on the next downstream packet and so on, forming a dependency chain. If this dependency chain forms a cycle, we have a network-level deadlock (Fig. 2(a)).

III. Motivation

A. Rarity of Protocol-level Deadlocks

We run PARSEC [6] and Splash-2 [65] applications on a mesh network to determine the frequency of protocol-level deadlocks. We use *MOESI Hammer* coherence protocol, and force all message types to be in the same class (i.e., no VNs). We use only one VC to increase the likelihood of a protocol-level deadlock. SPIN [55] is used to resolve only network-level deadlocks and we test the network with zero and eight faulty links. All tested applications complete successfully in the network with no faulty links. Protocol-level deadlocks only occur when 8 links are randomly removed for `FFT`. Thus the probability of protocol-level deadlock occurring is rare, which is consistent with prior work [60]; this motivates the need for a deadlock solution that does not impose significant overheads.

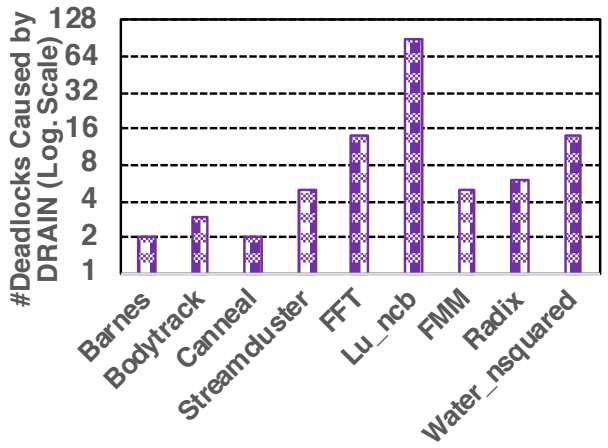


Fig. 3: Number of deadlocks generated by DRAIN [46] due to the draining/misrouting all packets in the network.

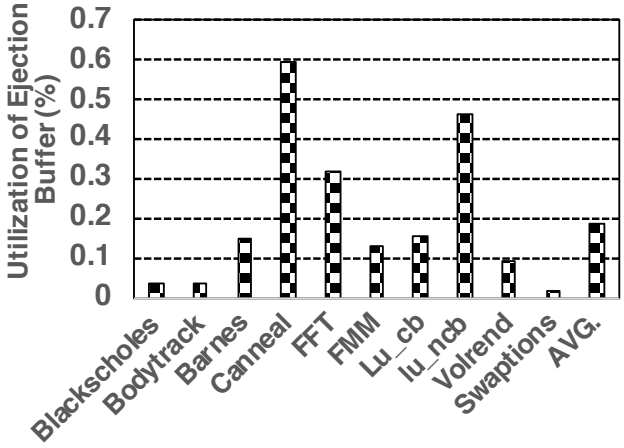


Fig. 4: Average utilization of ejection queue; utilization of the ejection queue is negligible. Note: The ejection queue holds 1-packet.

B. Performance Degradation of Misrouting

Many prior methods use misrouting to resolve deadlocks [24], [37], [43], [46], [64]. For example, SWAP [48] makes forward progress for one blocked packet at the cost of misrouting another. These misrouted packets may degrade performance, driving up the worst-case latency. We test this by determining how sensitive the performance of SWAP and SPIN are to increased congestion and deadlocks. To induce more deadlocks and congestion, we introduce eight faulty links within the network and use a 3-cycle router. For the more deadlock-prone configurations, we observe that both SWAP and SPIN increase 99^{th} percentile tail latency (i.e., worst-case latency) by 10× and 11× compared to a network with zero faults. Similar results for both schemes suggest that SWAP's misrouting incurs almost the same overhead as detection, synchronization and recovery methods seen in SPIN; SWAP increases tail latency because as congestion increases, it misroutes more packets–5.8× compared to 1-cycle router latency and no faulty links. To avoid this penalty, a solution should handle deadlocks without misrouting packets.

DRAIN [46] is another example which uses misrouting to resolve deadlocks. It infrequently forces all packets in the network to move. However unlike SWAP, which misroutes one packet at a time, all packets in the network may experience misrouting simultaneously. This type of misrouting may induce deadlocks in the network. As shown in Fig. 3(a), DRAIN introduces deadlocks for some benchmarks. Since draining is infrequent, these deadlocked packets may remain in the network for a considerable time before being resolved by a subsequent movement, thereby increasing worst-case latency. Note that network-level deadlocks do not occur for the benchmarks shown in Fig. 3 when a routing algorithm that does not guarantee deadlock freedom is used [46]. In general, if a deadlocked packet requires N-1 drains to resolve the deadlock cycle, the worst-case latency of the packet would be N×DRAIN period (i.e., 64K cycles).

C. Minimum Buffer Space Required

The minimum amount of buffer space required is dependent upon the number of VNs which is decided by the cache coherence protocol to maintain correctness. SPIN [55] and SWAP [48] only resolve network-level deadlocks and so they need multiple VNs to provide protocol-level deadlock freedom. DRAIN [46] can provide both network- and protocol-level deadlock freedom using no VNs; however, for DRAIN, eliminating VNs does not minimize the amount of buffer space required. To handle protocol-level deadlocks, DRAIN relies on an assumption that packets of one message class cannot be allowed to use up all of the resources (i.e., buffers) within the NoC. To satisfy this assumption, the number of *miss status handling registers (MSHRs)* should be relatively low compared to the amount of buffer space in the NoC, restricting the number of packets per message class. The minimum amount of buffer space while using no VNs must satisfy the worst-case scenario, i.e., every MSHR entry in the system is a store miss that needs to invalidate all L1 caches. Note that the store miss leads to $\#Cores - 1$ invalidation packets that all need to reside in the network. Therefore, the amount of buffer space needed to satisfy DRAIN's assumption is $\#MSHRs_{Entries} \times \#Cores \times (\#Cores - 1)$. DRAIN would either have to use multiple VNs to maintain protocol-level deadlock freedom or use the large amount of buffer space required when using no VNs. In the case of using no VNs, assuming 64 cores and 8 MSHRs in each core, the amount of required buffer space would be 32K buffer slots. However, this value would be less than 2K buffer slots while using multiple VNs. Therefore, even though DRAIN can guarantee protocol-level deadlock freedom without VNs, the buffer overhead required to do so is much more than it would be with VNs.To make a NoC work with no VNs, an efficient solution would need to handle both protocol- and network-level deadlocks while minimizing the amount of buffer space.

D. Ejection Queue Underutilization

Prior work has shown that the average utilization of VCs is low since the NoC is overprovisioned to handle peak loads [25]–[29], [43], [60]. Underutilization is made worse by the necessity of having multiple VNs for protocol-level deadlock avoidance. Recall that NI buffers (ejection and injection queues) and inter-router input buffers (e.g., $South$ buffer)

must use separate VNs per message class (Sec. II). Instead of devoting additional resources to handle deadlock issues which would exacerbate this underutilization, we aim to repurpose existing resources to solve deadlocks and save power and area. The NI ejection queue acts as a good candidate for temporary packet storage as its utilization is significantly low as shown in Fig. 4. The reason that ejection queue is significantly underutilized is that VCs of the ejection queue are only utilized by packets ejecting at that particular router. Further, ejection queues are generally drained almost immediately since the destination nodes are not typically busy. We observe that for PARSEC and SPLASH, 99.917% of the ejection packets encounter an empty ejection queue (with an ejection queue size of 1-packet).

IV. Pitstop

A. High-level Idea

The goal of this work is to design a low-cost solution to eliminate deadlocks in the unlikely event that they happen. We propose an inexpensive scheme, Pitstop, that inserts a bubble (an empty buffer slot) into a deadlocked router similar to Bubble Flow Control [11], [14], [53], [56]; however, Pitstop is not limited to ring topologies. As noted in Sec. III-D, the ejection queue in the NI is dramatically underutilized; thus, the ejection queue provides plenty of empty buffer slots (bubbles) to be placed in the deadlock chain. The bubble is inserted by swapping the empty slot in the ejection queue with the blocked packet in the router buffer. Think of it as the blocked packet taking a Pitstop in the NI which frees a buffer slot in the router.[1] Deadlock is broken after transferring the blocked packet to the NI. Pitstop adds low-complexity logic that ensures that the blocked packet makes forward progress through a bypass mechanism, hopping from NI to NI. Each router in the network is checked for blocked packets (either due to deadlock or congestion) in a statically determined order. If deadlock appears anywhere within the network, a bubble will eventually be placed within a router in the deadlock chain, resolving the deadlock.

Walk-through Example: Figs. 5 and 6 demonstrate how Pitstop resolves network- and protocol-level deadlocks. Starting with a network-level deadlock example (Fig. 5(a)), assume Pitstop starts with router R_0 and checks its $South$ input buffer. Due to the deadlock, the packet A_1 held in the $South$ input buffer cannot continue its path. Pitstop transfers this blocked packet to the ejection queue of R_0 (assuming the ejection queue has enough space) which frees the $South$ input buffer and breaks the network-level deadlock (Fig. 5(b)). The previously blocked packet A_1 now residing in the NI of R_0 is then transferred to the ejection queue of the downstream router (R_1) provided that the ejection queue is not full (Fig. 5(c)). Packet A_1 now resides in router R_1's NI, and can be re-injected through the injection queue to continue its path (Fig. 5 (d)).[2]

[1]Like a Formula 1 car, the packet is temporarily exiting the ring (track) it is on, which will enhance the movement in the network.

[2]Qn 1, Qn 2, Qn 3 discuss when injection/ejection queues are full.

After resolving the deadlock and allowing the blocked packet to make forward progress, Pitstop moves to the next adjacent router to resolve any potential deadlocks.

Protocol-level deadlocks are resolved similarly. The injection and ejection queues contain a separate queue per message class. In Fig. 6(a), the NI of R_1 and R_2 cannot receive A_1 and A_3 responses as they are blocked by A_2 and A_4 requests. This is a cyclic dependency and a protocol-level deadlock. At any given time, Pitstop can transfer one blocked packet for each message type. Assuming Pitstop starts with R_0's $South$ input buffer and R_1's $West$ input buffer, blocked packets of different message types A_1 and A_2 are transferred to their respective ejection queues (Fig. 6(b)). A_1 and A_2 bypass the current routers and are sent to the ejection queue of R_1 and R_2 (Fig. 6(c)), making forward progress and breaking the deadlock since all request and response packets have reached their destinations. All bypassed packets reside in the corresponding queue of the NIs (Fig. 6(c)). Compared to prior work, both types of deadlocks are resolved simultaneously without misrouting or using VNs by applying Pitstop router by router. The order that Pitstop is applied to each router is determined statically, adding minimal hardware complexity. Note: Blocked packets are not necessarily deadlocked; more likely they are blocked from congestion in which case Pitstop helps congested packets make forward progress, reducing average packet latency.

B. Assumptions and Definitions

Pitstop makes the following assumptions:

1) The topology stays connected even in the presence of faulty links. There are no isolated routers in the NoC and they all are connected through bidirectional links. In the presence of faulty links, all packets can still reach the desired router at the cost of increased hop count. Should a fault occur on one of the unidirectional links, we assume that both unidirectional links are faulty. Due to the connectivity assumption, constructing a spanning tree covering all bidirectional links is possible.
2) Faults only occur in the links and not in the routers. This is a fair assumption as occurrences of link failures are far more common than routers failures [33], [38].

With the these assumptions, we guarantee a network- and protocol-level deadlock free NoC with no VNs.

Definitions: We use the following terminology:

- **Root Router:** A router which examines all its input buffers sequentially, including its injection queue, to check if there is a blocked packet. *There can be multiple root routers in the network; however, each root router is statically associated with one message class* (e.g., N message types correspond to N different *root routers*). In Fig. 6, the *root routers* are R_0 for response messages and R_1 for request messages.
- **Golden Packet:** The packet blocked due to congestion or deadlock, picked by the *root router* to make forward progress. *Only one golden packet of each message type is*

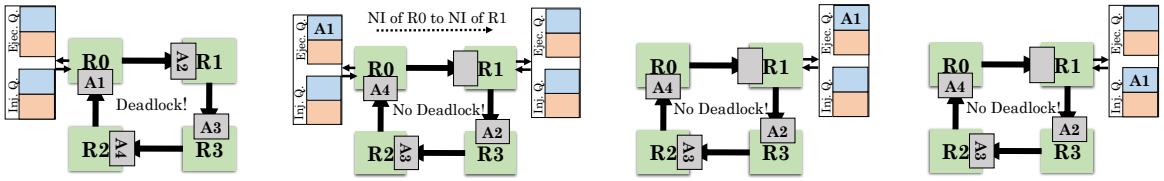


Fig. 5: Pitstop solution to handle network-level deadlock, (a) A cyclic dependency between A_1, A_2, A_3, and A_4 packets; Pitstop examines south input buffer of R_0, (b) Pitstop transfers A_1 (i.e., a blocked packet) to the ejection queue of R_0, breaking deadlock cycle, (c) Pitstop bypasses R_0 and sends A_1 to the ejection queue of R_1, making one hop progress for A_1, (d) Pitstop re-injects A_1 to the injection queue of R_1 to continue its path.

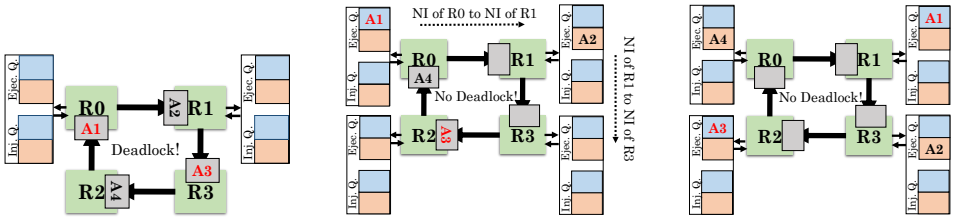


Fig. 6: Pitstop solution to handle protocol-level deadlock, (a) Network interface of R_1 and R_2 cannot receive A_1 and A_3 responses, blocked by A_2 and A_4 requests; Pitstop examines south input buffer of R_0 and west input buffer of R_1, (b) Pitstop transfers A_1 response and A_2 request (blocked packets) to the ejection queue of R_0 (i.e., blue queue) and R_1 (i.e., orange queue); now A_3 response and A_4 request can reach R_2 and R_0, (c) Pitstop bypasses R_0 and R_1 by sending A_1 response to the ejection queue of R_1 (i.e., blue queue) and A_2 request to the ejection queue of R_3 (i.e., orange queue); now, all the requests and responses packets have reached their destinations successfully.

allowed in the network at any time. In Fig. 6, the *golden packets* are A_1 and A_2 since they are of different types.

- **Pitstop Procedure:** The sequence of steps involved in transferring the *golden packet* in the *root router* to its own ejection queue (first step) and then to the ejection queue of the downstream router (second step), followed by re-injecting it to its injection queue (third step). If the *golden packet* resides in the injection queue, the first step is skipped.
- **Leaf Router:** A router at the end of the *Pitstop procedure*, which is the destination of the *golden packet* or re-injects the packet to its injection queue. In Fig. 6, the *leaf routers* are R_1 and R_3.
- **Chain Length:** The number of routers that a *golden packet* traverses to reach the *leaf router*. In Fig. 6, the chain length is one for both *pitstop procedures* (R_0 to R_1 and R_1 to R_3). The maximum value of the *chain length* is equivalent to the maximum number of hops in a topology.

C. Router Microarchitecture

Fig. 7 illustrates the Pitstop router microarchitecture and its network interface. Compared to a generic NoC (Sec. II), Pitstop does not use any VNs for the $North$, $South$, $East$, and $West$ buffers; however, it still has one queue per message type in the injection and ejection buffers. Pitstop adds several multiplexers, demultiplexers, and a management unit to guarantee forward progress for a blocked packet. This management unit is responsible for controlling the *pitstop procedure* and coordinating the handshaking signals (i.e., *ready, request, done, and root*), between the routers in the *chain* and the control signals for all multiplexers. All handshaking signals are $\lceil log_2(N) \rceil$ bits, where N is the number of message types. The management unit also records if this router is currently the *root router*.

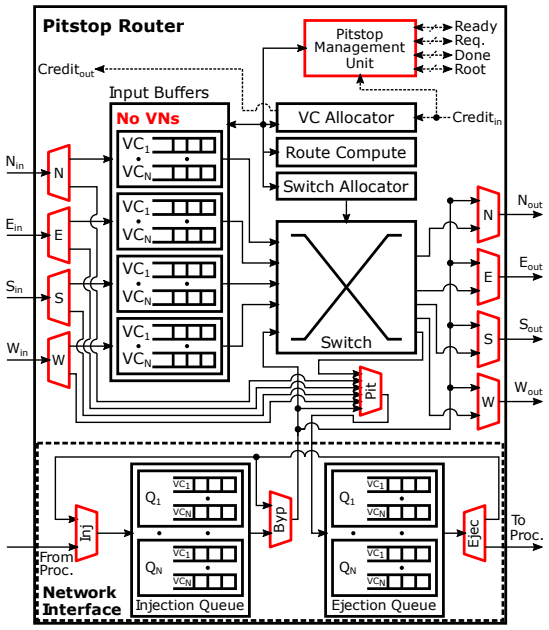


Fig. 7: Pitstop architecture; changes to support Pitstop are shown in red. The $South$, $North$, $East$, and $West$ inputs have no VNs; injection and ejection buffers have separate queue per message type.

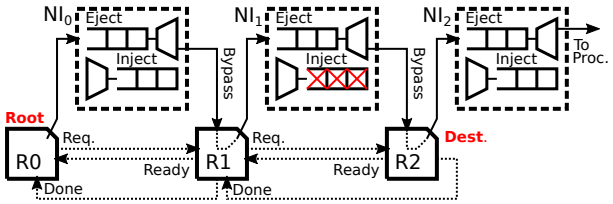


Fig. 8: Example of NI-NI traversals. Router R_0 is *root router* and R_2 is the destination of the packet. Injection queue of R_1 is full so another NI-NI traversal is triggered.

D. Detailed Algorithm

1) Selecting Root Routers: As mentioned in Sec. IV-B, Pitstop allows one root router per message type in the network at any time; in other words, multiple golden packets are allowed in the network–one per message type. In a network with N message types, a single router could be root for all N types at the same time. Allowing multiple root routers in the NoC makes Pitstop capable of resolving multiple protocol-level deadlocks simultaneously. To guarantee correctness, Pitstop gives all the routers the chance to be a *root router* for each message type, regardless of whether deadlock or congestion occurs. We define a pre-determined order of the routers to reduce the hardware complexity of switching roots. *Root routers* for each message class are distributed along the pre-determined path. Root routers are selected contiguously (i.e., switching a root to the next adjacent router within the network). The root switch path must visit all routers at least once to make a complete pass. A *root router* for one message type can catch up with and go ahead of another *root router* along the *root* switching path if its *pitstop procedure* completes faster. A *root router* examines each input buffer in a round-robin fashion (e.g., $South$, $North$, $East$, $West$, and $injection$ buffers) for its respective message type. When no *golden packet* of that type exists in the router, its done signal is set. The *root* switching procedure is coordinated through each router's Pitstop management unit; handshaking signal are sent through the extra sideband wires shown in Fig. 7. Root status is passed to the next adjacent router in the path by enabling the root signal after all input buffers have been checked. This procedure continues indefinitely through every router. Deadlocked or congested packets of each message type eventually encounter a *pitstop procedure* and will make forward progress.

2) Enabling a Pitstop Procedure: Based on the following conditions, Pitstop management unit begins a *Pitstop procedure*: (1) the routers which need to start the procedure must be root routers, (2) a root router must receive its *done* signal before starting another procedure. At any given moment in time, each message type can only have one golden packet in the network, (3) there are no available slots in the downstream input buffer for the current golden packet. No available buffer slots indicate either congestion or deadlock, and (4) the golden packet's destination should not be its *root router*. If all of these conditions are satisfied, a root router can start a Pitstop procedure to make forward progress for its golden packet. If the *pitstop procedure* is enabled for a *root router*, the *golden packet* is transferred via the switch to the ejection queue (into its respective queue) of a *root router* through the Pit multiplexer.[3] Recall that if the *golden packet* resides in the injection queue, this step is skipped. In cases where one physical router is simultaneously the root for multiple message types, golden packets must be transferred serially. A *root router* subsequently sends a *request* signal to the downstream router indicating that a *golden packet* needs to be sent to the ejection queue of that router. The downstream router responds to that request by sending a *ready* signal to the *root router* if its ejection queue is not full.[4] After receiving the *ready* signal, the *golden packet* bypasses the *root router* and is written to the ejection queue of the downstream router through the bypass (Byp) multiplexer followed by one of the appropriate multiplexers and demultiplexers in the output and input stages (e.g., N multiplexer and S demultiplexer, respectively). From the input demultiplexer, the packet goes through the Pit multiplexer into the next ejection queue. We called this traversal an *NI-NI traversal*. Since there is a *root router* for each message type, multiple *pitstop procedures* may happen concurrently throughout the network (e.g., Fig. 6 where R_0 and R_1 are different *root routers*). Each root router can pass root status to the next router in sequence without waiting for other root routers to complete their pitstop procedures; no global coordination or handshaking is required.

If there is space in the injection queue of the downstream router after the NI-NI traversal, the golden packet is written to the respective injection queue through the Inj multiplexer to continue on its path; this router constitutes the leaf router. The leaf router then sends a *done* signal to the requester, which is the *root router* in this case. The requester does not have to be the *root router* (in cases where chain length is greater than one), but the *done* signal always propagates back to the corresponding *root router* of the message type.

Qn1: What if the injection queue is full? If the injection queue of the current router is full, the current router sends a *request* signal to the downstream router and sends the packet to the ejection queue of that router (through the bypass mechanism) after receiving the *ready* signal. Fig. 8 shows an example where R_0 is a root router for a message type initiating a Pitstop and R_2 is the destination of the *golden packet*. The *golden packet* goes through an NI-NI traversal from NI_0 of R_0 to NI_1 of R_1. If NI_1's injection queue is not full, the *golden packet* would be re-injected; however in this case, the injection queue is full. R_1 sends a request to R_2 to initiate another NI-NI traversal of the *golden packet*, which then is ejected to the processor. A *golden packet* reaches a *leaf router* by either experiencing multiple *NI-NI traversals* or by moving into an injection queue through the Inj multiplexer. For both cases, a *leaf router* sends a *done* signal to its requester. Each router receiving a *done* signal propagates it until it is received by the corresponding *root router*.

[3]See Qn 2 for when ejection queue of a root router is full.
[4]See Qn 3 for when ejection queue of a downstream router is full.

The inject (Inj) multiplexer is also responsible for allocating injected packets from the processor to the injection queue. If the two inputs of the multiplexer are ready at the same time, priority is given to the *golden packet*. Note that when a *golden packet* resides in the ejection queue of a *root router*, it is always sent to the downstream router by the bypass (Byp) multiplexer as mentioned earlier.

Qn2: What if the ejection queue of a root router is full? If a *golden packet* resides in one of the input buffers of a *root router*, it needs to wait until either the ejection queue (in the corresponding queue) has free space (the *golden packet* is blocked due to a deadlock) or the downstream buffer has free space. In either case, Pitstop guarantees correctness as the congestion gets resolved or the ejection queue eventually delivers one packet to the processor making space for the *golden packet*. Ejection queues will never be full indefinitely due to deadlock since we assume separate ejection queues per message class. There will always be at least one *sink* message class (e.g., response messages) that corresponds to the end of a communication transaction so ejection queues for sink message types can always be drained. Although ejection queues for request messages may stall and fill up due to the cache stalling for a response deadlocked elsewhere in the NoC, eventually the response message that will unstall the node will arrive and requests in the ejection queue can be consumed, freeing up request buffer slots. Since Pitstop allows one golden packet of each message type at any time, ejection queues are guaranteed to eventually free up.

Qn3: What if a golden packet resides in the ejection queue of one router waiting to receive ready signal from the downstream router; however, the ejection queue of downstream router is full? There are two possible scenarios. Scenario (1) is that a *golden packet* must wait until the ejection queue of the downstream router finds free space. Once there is free space, the downstream router sends the *ready* signal, and Pitstop successfully bypasses the packet. Recall that we assume each message type is allocated to a single queue within the NI. Ejection queues of sink message types (e.g., response messages) eventually have free buffer slots as their messages can always be consumed. Request type ejection queues may stall, but not indefinitely, since Pitstop has *root routers* for each message type and ensures response messages that will unstall the node will eventually arrive resulting in consumption of the request messages. Note that *golden packets* of each message type will not collide since they must be allocated to the injection/ejection queues of their own type. *Golden packets* all operate independently. Scenario (2) is that the injection queue of a non-root router gets free space before receiving the *ready* signal from the downstream router. Recall that Pitstop first looks at the injection queue (for the non-root routers) and if it is full, sends a *request* signal. In this scenario, Pitstop disables the *request* signal as there is no longer the need to bypass the *golden packet* and instead transfers it to the injection queue where it can continue on its path.

3) Maintaining the Chain: Each router that receives a *golden packet* increases the *chain length* for its message type by one, where the first node is the *root router* and the last node is the *leaf router*. Routers only need to store the requester of the received *request* signal for its message type. The *done* signal from the *leaf router* is propagated back to the *root router* by checking each router's requester. Each router requires storage to keep track of a requester for each message type. In a mesh, this information is encoded with two bits (indicating the direction of the requestor). These two bits determine how to propagate the done signals backwards towards the root.

Qn4: Does Pitstop guarantee that the chain terminates? Yes: the chain length is the number of routers that a golden packet traverses to reach its leaf router. The worst-case scenario occurs when all the injection queues of the routers located in the chain are full causing the golden packet to experience multiple NI-NI traversals until it reaches its destination. In this scenario, the maximum value of the chain length is equivalent to the maximum number of hops in the topology. Therefore, since Pitstop uses minimal routing and does not misroute any packets, no livelock can occur and makes revisiting a router impossible. Hence, Pitstop guarantees that the chain length is finite. It is possible to always send the golden packet to the destination through NI-NI traversals but this would cause longer chain lengths. A longer chain length increases the time each router stays the root for which then increases the time for the root status to be passed to the next adjacent router, and hence takes longer to resolve deadlocks elsewhere in the network.

4) Pitstop Latency and Flow Control: Pitstop procedure takes a minimum of four cycles assuming the ejection queues of the *root router* and downstream router are not full. Pitstop would transfer the *golden packet* to the ejection queue and send a *request* signal to the downstream router in the first cycle. The downstream router returns the *ready* signal in the second cycle and then the *golden packet* is transferred to the downstream router in the third cycle. In the fourth cycle, the downstream router transfers the *golden packet* from the ejection queue to the injection queue and sends the *done* signal to the *root router*. The *golden packet* is chosen since it was blocked, either due to deadlock or congestion. Although it may seem that the *pitstop procedure* incurs latency, it in fact decreases packet latency by freeing the blocked packet to make forward progress.

Qn5: Does Pitstop support wormhole flow control? Yes: Pitstop supports both wormhole and virtual cut-through (VCT) flow controls. Unlike the prior work where packet misrouting requires packet truncation to support wormhole [46], [48], [55], Pitstop does not misroute packets, and thus does not impose any overheads to support wormhole flow control. The only restriction is placed on the queues in the NI. Since multiple ports now share the injection and ejection queue, the full packet has to be received before giving access to another packet (like VCT) to prevent interleaving of flits from different packets.

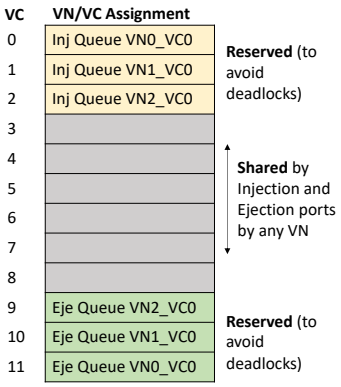


Fig. 9: Hypothetical shared injection/ejection buffer design with VCs reserved for deadlock avoidance assuming three message classes in protocol.

Qn6: Are ejection buffers common? Pitstop assumes that NI consists of injection and ejection buffers. This assumption is standard in most the academic papers [10], [16], [46], [50], [60], prototypes [4], [17], [58], [63] and commercial implementations [9], [36]. For example, SCORPIO [17] and IBM BLUE GENE/Q [9] use separate FIFOs for the injection and ejection buffers–one per message type. Ejection buffers at NIs are needed since an endpoint node/IP might be stalled and cannot receive a packet [21], [61]. In the event that IP blocks are in a different clock domain than the NoC, ejection buffers are often implemented as bisynchronous FIFOs to serve as clock boundary crossing buffers [36], [57], [58]. In this case, while there will be some added latency in the credit return loop due to the need for the credit signal to be resynchronized into the NoC's clock domain, the same credit-based backpressure mechanisms will be used as in other input buffers. Thus Pitstop will still work correctly, although clock crossings might slow down the Pitstop procedure. Considering that deadlocks are rare [46], [55], [60] and the fraction of packets marked as golden is low (Fig. 12), clock crossing should not have a large performance impact.

Qn7: Does Pitstop work with alternative ejection buffer designs? Several works investigate the use of shared buffers to improve buffer utilization [29], [35], [45], [54], [62]. Although these papers do not share the ejection buffer with other ports, Pitstop will work correctly should ejection and injection packets share the same physical buffer consisting of separate queues per message type. For correctness in such designs, it should be impossible for the injection packets to occupy all the shared buffer slots, leaving no space for the ejection packets. Thus at least one slot of the shared buffer (per message type) should be dedicated to ejection packets. Fig. 9 shows an example of such a shared buffer design operating with with 3 message classes (3 VNs). After some slots (i.e., VCs) are reserved for correctness, there are a number of slots that can be shared. In a worst-case scenario, a golden packet could reach its destination by experiencing multiple NI-NI traversals using the reserved ejection slots; therefore the proof in Sec. IV-E would still be valid for such scenarios. We provide quantitative results for this scenario in Sec. VI.

E. Proof of Correctness

Lemma 1. *In a deadlocked path of length k, at most $k-1$ NI-NI traversals using the Pitstop procedure are needed to resolve the deadlock.*

Proof. Traversing each hop leads to forward progress in minimal routing. Having k NI-NI traversals would return the golden packet back to its initial position – contradicting the definition of minimal routing. In contrast, experiencing at most $k-1$ NI-NI traversals makes a golden packet either exit the deadlocked path or reach its destination which is located on the deadlocked path, thus resolving the deadlocks. □

Lemma 2. *As there will eventually be free space in the corresponding ejection queue, a root router can successfully complete the Pitstop procedure.*

Proof. The Pitstop procedure relies on access to ejection queues; if we can guarantee that a deadlock never occurs in the ejection queues then there will always be at least one free space at the ejection queue of a root router that can be used by the Pitstop procedure to make forward progress for a golden packet. Each message class is assigned to a different queue in the ejection buffer. The ejection queue for sink type messages (e.g., response type) can always be consumed; the ejection queue for sink types will always empty itself. In contrast, ejection queues for request types may stall and fill up due to the cache stalling for a response deadlocked elsewhere in the network. However since Pitstop supports golden packets for each message type, eventually the deadlocked response packet will be received and un-stall the node (Lemma 1). Receipt of the response, will allow the processor to resume processing requests, which frees up ejection queue space for that message type. Therefore all ejection queues will eventually have free space. □

Lemma 3. *As all network routers have the chance to become root routers to enable the Pitstop procedure, freedom from network- and protocol-level deadlocks is guaranteed.*

Proof. Since applying the Pitstop procedure resolves a deadlock (Lemma 1) and the ejection queues are guaranteed to be available eventually (Lemma 2) then if each router gets a chance to start a Pitstop procedure, forward progress is guaranteed for any deadlocked packet. Pitstop gives each router a chance to start a Pitstop procedure for each input buffer and each message type by passing the root signals to all routers repeatedly. As a result, both network- and protocol-level deadlocks are guaranteed to be resolved. □

Qn8: What if packets of one message class use up all of the buffers in the network preventing packets of other message classes to be injected to the NoC? Unlike prior work [46] which assumes it is impossible for packets of one message class to use up all of the resources (Sec. III-C), Pitstop is capable of handling this situation thanks to its *NI-NI traversals*. Suppose all the NoC resources are occupied by request packets, so response packets cannot be injected into the NoC. Since there is always one root router of each

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 and 8×8
Evaluated Schemes	Escape VC [18], SWAP [48] (swap duty: 1 cycle) SPIN [55] (detection threshold: 128 cycles) DRAIN [46] (DRAIN period: 64K cycles)
Routing Algorithm	SWAP, SPIN, DRAIN, Pitstop (Fully adaptive routing) Escape VC: XY (Up*/Down*) within escape VC and fully adaptive within other VCs for regular (irregular) topology
Router Latency	1-cycle and 3-cycle
Number of VNETs	0-VN (Pitstop) and 6-VN (escape VC, SPIN, SWAP, DRAIN)
Number of VCs	Pitstop (2, 3, 4, 5) per input buffer Escape VC, SPIN, SWAP, DRAIN (2 VCs per VN)
Buffer Size	5-flit
Link Bandwidth	128 bits/cycle
Flow Control	VCT – Single packet per VC
Number of Faults	0, 8 (PARSEC, SPLASH-2) and 0, 12 (Synthetic traffic)
Synthetic traffic	Uniform, Transpose, and Shuffle – Mix of 1-flit and 5-flit

message type at any time and a root router examines all its input buffers, response packets can reach their destinations by experiencing multiple *NI-NI traversals*; the root router for a response packet (i.e., golden packet) enables the *Pitstop procedure* and sends the packet to the ejection queue of the downstream router (into its respective queue). Since all the buffers in the network are full with the request packets, the response packet experiences multiple *NI-NI traversals* until it reaches its destination. Receipt of the response results in consumption of the request packets, which frees up the ejection queue space for the request type.

Qn9: Is it possible for a golden packet to be transferred indefinitely between the ejection and injection queues of one router due to a deadlock? No: A root router always sends a golden packet to the NI of the downstream router. In other words, a root router never uses the inject (Inj) multiplexer (Fig. 7) to transfer a golden packet. This multiplexer is used when a router receives a golden packet and wants to transfer it to its injection queue and send the done signal. In this case, it might be possible that the packet needs to experience another Pitstop procedure if there is a deadlock in the router that it arrived at. The Pitstop procedure would then be executed when that router becomes a root router.

Qn10: Is livelock possible due to a router receiving unlimited request signals from other routers? No: A router cannot receive unlimited *request* signals from its neighbors since *root* status round-robins through all routers, giving each router one chance to be a root router per message type. Thus, while a router is the root for a message type, it is impossible to receive any request signal of that message type – allowing the ejection packet of that message type (currently residing in $North$, $South$, $East$, or $West$) to reach the ejection queue.

V. Methodology

We evaluate Pitstop using full-system *gem5* [7] with the *Garnet2.0* [1] network model and the *Ruby* memory model and run PARSEC [6] and Splash-2 [65] applications. Tab. II lists the configuration used.

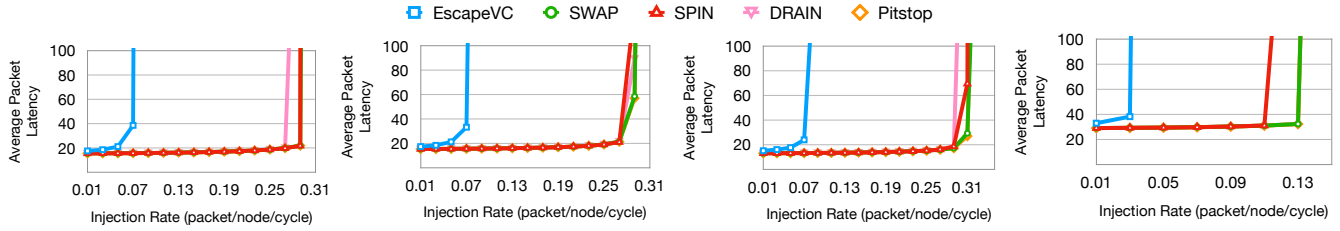
Topology: Pitstop is evaluated using both regular (mesh) and irregular topologies. Irregular topologies are derived by randomly eliminating links from a mesh network, which emerge in different scenarios such as a faulty networks [41], [52] or power-gated techniques [10], [38], [52]. Since Escape VC [18]–[20] applies turn restrictions on the escape VC, it cannot be used with an irregular topology. Hence, we augment Escape VC with a spanning tree based up*/down* routing [59]. Note that we assume information of faulty links and the routing path is based on offline computation [41], [51].

VI. Evaluation

A. Performance

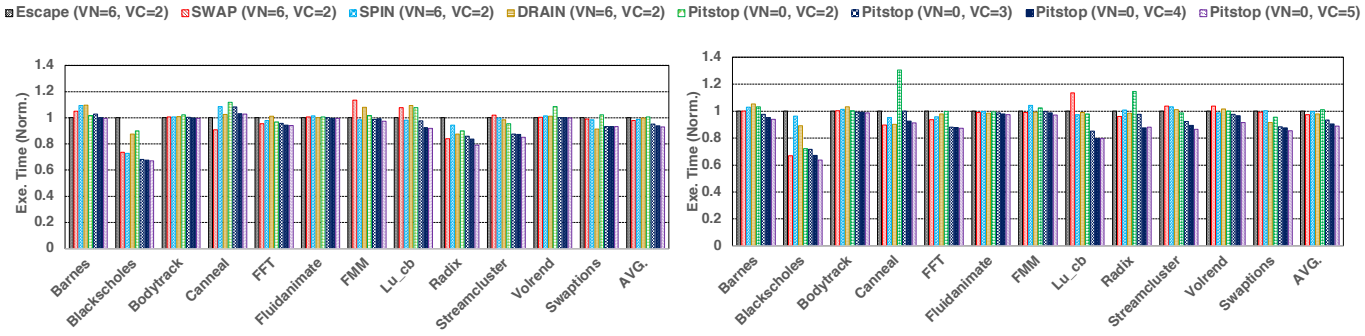
Synthetic Traffic: Fig. 10(a-c) shows performance of the existing techniques and Pitstop for various synthetic traffic with two VCs and no faulty links. All techniques saturate at a higher point compared to Escape VC due to the ability to support fully adaptive routing which lowers congestion within the network. Pitstop has the same saturation throughput as SPIN and SWAP. The throughput loss of DRAIN for *Uniform* and *shuffle* traffics comes from deadlocks occurring at higher injection rates as the deadlocked packets remain in the network till the next DRAIN period(s). No deadlocks occur for the *Transpose* traffic. Fig. 10(d) examines the sensitivity of saturation throughput to irregular topologies and number of router pipeline stages. The irregular topology is simulated by injecting 12 random faulty links within a single mesh network. Each router is configured to have three pipeline stages. The saturation throughput is lower for the faulty network for all techniques due to higher congestion compared to the fault-free topology. Escape VC has the lowest throughput due to the use of up*/down* routing within the escape VC. Pitstop matches SWAP and DRAIN while SPIN's suffers from the additional latency in deadlock detection and global synchronization. The additional faulty links within the network increase the occurrence of congestion, exacerbating the added latency in SPIN's deadlock mechanism.

PARSEC and SPLASH-2 Benchmarks: We compare Pitstop's execution time to prior techniques for zero and eight faults (Fig. 11). We test Pitstop using two to five VCs. For zero faults in the network, Pitstop with three to five VCs improves execution time by 4.8%, 6.2%, and 7%, respectively over Escape VC. Pitstop using two VCs degrades performance by a negligible 0.7% over Escape VC while using *6× fewer VCs compared to all the existing schemes*. Application traffic does not fully use NoC resources due to typically low injection rates. This study brings up the rarity of deadlocks again while demanding a network free of both kinds of deadlocks. Pitstop guarantees both kinds of deadlock freedom while eliminating all the costly VNs and getting better VC utilization. To increase the probability of deadlocks, we inject eight faulty links to the NoC. Pitstop has greater performance improvements over the prior work for an irregular topology. Compared to Escape VC, Pitstop using three to five VCs improve execution time by 6.5%, 9.4%, and 10.9%, while Pitstop using two VCs degrades performance by only 1.1%. The faulty network increases congestion and the likelihood of



(a) Uniform with no faults (b) Transpose with no faults (c) Shuffle with no faults (d) Uniform with 12 faults

Fig. 10: Performance of the existing techniques and Pitstop for synthetic traffic with a mix of one and four flits in an 8×8 regular mesh topology (a-c) using 1-cycle router latency and an irregular topology (d) using 3-cycle router latency.



(a) Fault 0 execution time (b) Fault 8 execution time

Fig. 11: Execution time (normalized to Escape VC) for zero faults and eight faults.

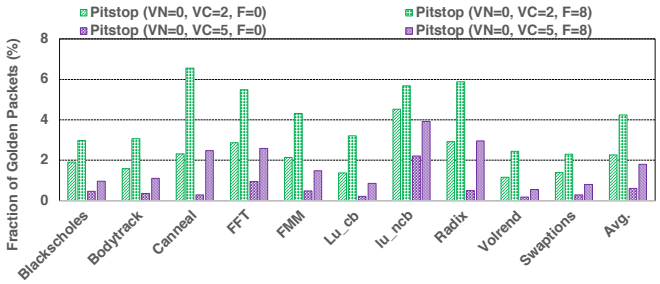


Fig. 12: Fraction of *golden packets* with zero faults (F=0) and a network with eight faulty links (F=8) using Pitstop with two and five VCs.

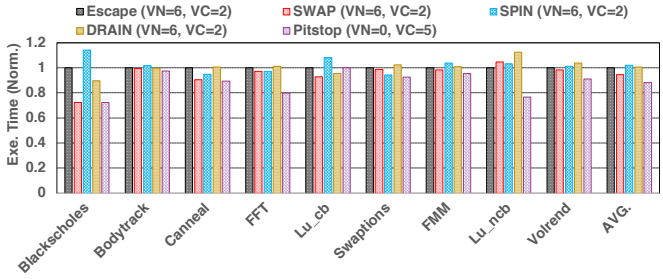


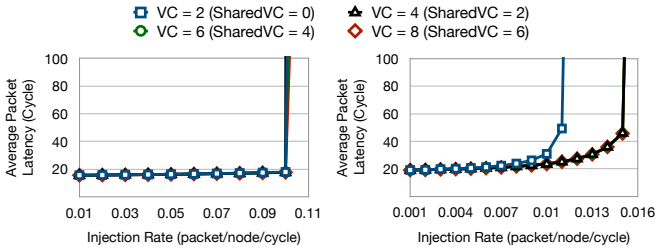
Fig. 13: Execution time (normalized to Escape VC) for eight faults and 3-cycle router latency.

deadlock highlighting the effectiveness of the Pitstop design. A larger fraction of packets become golden packets in the faulty network as shown in Fig. 12. More congested packets are making forward progress through the *NI-NI traversals* which improves performance.

Increasing the number of VCs in Pitstop improves performance for two primary reasons: (1) Pitstop has higher utilization of its VCs to reduce head-of-line blocking. Each message type in the prior schemes only has access to two VCs in each VN, whereas in Pitstop, all message types have access to all VCs. Note that there are no message types in Fig. 10 (no VNs) since cache coherency is not considered in synthetic traffic; that is why Pitstop does not improve performance in Fig. 10. (2) The *pitstop procedure* bypasses congested packets to make forward progress–decreasing packet latency. Note that even for the lowest average execution time (Pitstop (VN=0, VC=5)), Pitstop uses only five VCs per input buffer compared to 12 in the existing techniques.

To further stress the NoC, we also compare Pitstop (with five VCs) to the existing techniques in a faulty network using 3-cycle routers shown in Fig. 13. Pitstop improves execution time by 11.6%, 13.6%, 6%, and 12% compared to Escape VC, SPIN, SWAP, and DRAIN. The reason Pitstop beats SWAP is that SWAP increases the percentage of misrouted packets 5.8× compared to 1-cycle router latency and no faulty links.

Shared Buffer Design: In Sec. IV-D, we discuss the correctness of Pitstop under shared injection/ejection buffer design presented earlier in Fig.9. Fig. 14 presents synthetic traffic results for this design. Since traffic is synthetic, there are no protocol message classes, so the minimum number of VCs required for correct operation is two - one reserved for injection and one for ejection. The lines in the figure show latency as a function of increasing number of VCs in the NIC (i.e., increasing the available shared VCs). The NoC itself has only one VC under all conditions. Under uniform random traffic, we find that throughput is limited by the link bandwidth to/from the NIC; increasing the NIC injection and ejection buffers through sharing does not have an impact on



(a) Uniform (b) Hotspot

Fig. 14: Shared buffer design under (a) uniform and (b) hotspot traffic.

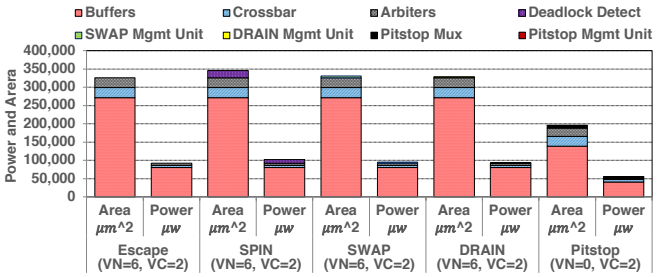


Fig. 15: Post Place-and-Route router area and power (28nm TSMC, 1GHz).

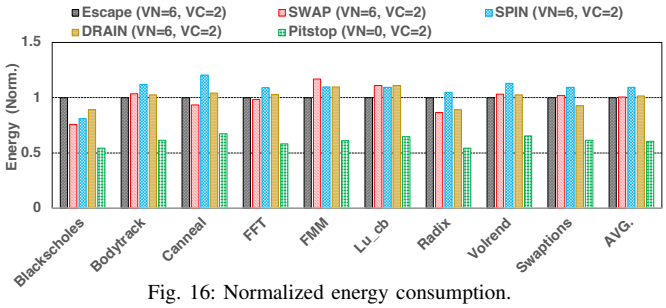


Fig. 16: Normalized energy consumption.

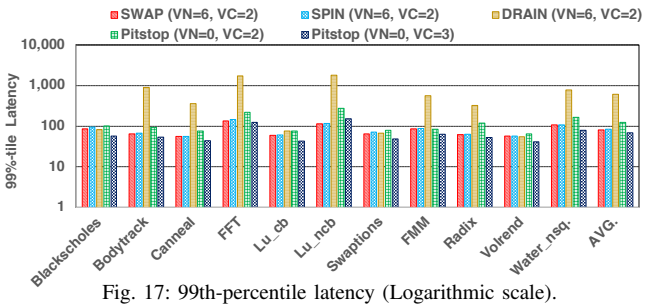


Fig. 17: 99th-percentile latency (Logarithmic scale).

throughput. For HotSpot traffic where all nodes send to the same destination, we find increasing the amount of shared VCs in the NIC does improve throughput due to credit turnaround time. Application-level results are omitted as we found no measurable performance impact from increasing the amount of shared VCs beyond the minimum number of slots for correctness (1 per VN for both injection and ejection). This is consistent with our observation of low VC utilization at the NIC for application traffic (Fig. 4) under nominal operation.

B. Area, Power and Energy

The number of required VNs is determined by the coherence protocol; since we use the *MOESI Hammer* protocol, the minimum number of required VNs to guarantee protocol-level deadlock freedom is 6. SWAP and SPIN only handle network-level deadlocks so they must use 6 VNs to avoid protocol-level deadlocks. DRAIN can handle protocol-level deadlocks but it needs to use multiple VNs to avoid the large amount of buffer space required when using no VNs (Sec. III-C). In contrast, Pitstop guarantees both network- and protocol-level deadlock freedom with no VNs while minimizing the amount of buffer space. Fig. 15 shows the static power and area breakdown of Pitstop (VN=0, VC=2) compared to Escape VC (VN=6, VC=2), SPIN (VN=6, VC=2), SWAP (VN=6, VC=2), and DRAIN (VN=6, VC=2). We implement them using open-source RTL [40] and synthesized and placed-and-routed using TSMC 28nm, targeting 1GHz with 1-cycle pipelines. Our calculations factor in hardware modifications in both the router and the NI. The additional hardware needed in SPIN, SWAP, and DRAIN imposes approximately 6%, 1.6%, and 0.8% overhead compared to Escape VC (VN=6, VC=2). Although Pitstop implements some additional hardware, we see power and area savings of 41% and 40% due to the decrease in total number of buffers by eliminating VNs. The Pitstop management unit and the added multiplexers/demultiplexers consume only ∼0.4% and ∼3.7% of Pitstop area.

Energy: Area of Pitstop is minimal; with abundant transistors, a more important metric is *energy*. Fig. 16 shows the energy of the different deadlock freedom methods normalized to Escape VC. By eliminating VNs (saving power) and getting better VC utilization (improving performance), Pitstop (VN=0, VC=2) reduces static energy consumption by 40%.

C. Sensitivity Study

Tail Latency: Fig. 17 shows the 99^{th} percentile tail latency (i.e., worst-case latency) of network packets for Pitstop using two to three VCs compared to prior work. Since Pitstop allows one root router for each message type in the network at any given time, we see improvements in tail latency using three VCs due to better utilization and greater reduction in head-of-line blocking. On average, Pitstop reduces tail latency by 18.5% and 15.2% as compared to SPIN and SWAP. Pitstop with two VCs sees higher tail latencies (46% increase over SPIN) due to all message types using two VCs. DRAIN introduces deadlocks for some benchmarks (e.g., `FFT`) which causes a significant increase in tail latency (Sec. III-B). However, it matches SPIN and SWAP for the rest (e.g., `Blackscholes`) in which deadlocks do not happen. On average, Pitstop with even two VCs sees significant reduction in tail latency compared to DRAIN.

Qn11: What if deadlocks are more frequent? Fig. 18 compares the performance of state-of-the-art schemes to West-first routing as deadlock rate increases. To increase the probability of a deadlock occurring, all techniques use 1 VC. We force the routing algorithms of the state-of-the-art schemes to make a clockwise cycle for *turn* packets causing deadlocks for all injection rates shown. Throughputs in SPIN and DRAIN are very low with high rates of deadlock since SPIN needs global synchronization and deadlock detection while DRAIN forces deadlocked packets to persist in the network until the next DRAIN period(s). SWAP and Pitstop achieve the

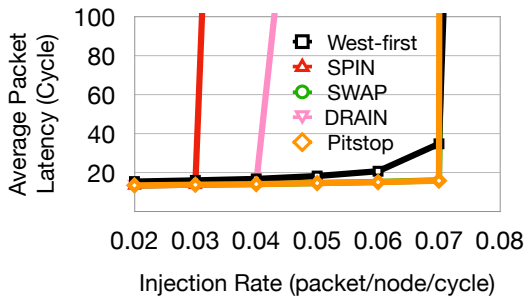


Fig. 18: Performance of the state-of-the-art schemes and Pitstop compared to West-first as the deadlock rate increases for bit-complement synthetic traffic; all techniques use 1VC.

same performance as West-first but West-first cannot support irregular topologies or faulty networks.

Qn12: Does Pitstop affect the critical path? Typically the critical path is dictated by the switch crossbar and the allocators [23], [40]. Pitstop does not make any modifications to these units as shown in Fig. 7; however some modifications (e.g., the mux denoted Pit in the figure) may increase the critical path. To calculate the critical path, we use the same placed-and-routed RTL implementation as in Sec. VI-B. Compared to Escape VC (VN=6, VC=2), our Post synthesis report shows that Pitstop (VN=0, VC=1) shortens the critical path by 31%. Although the Byp and Pit multiplexers increase the switch delay, eliminating VNs shortens the critical path. Recall that a router has to use six VNs for each input port just to guarantee protocol-level deadlock freedom for the MOESI Hammer protocol. The more VCs, the longer the critical path [8], [21]; thus, VC based approaches to avoiding deadlocks have a negative impact on the critical path. Since Pitstop is capable of handling both deadlocks with no VNs, the critical path is shortened. Thus, the increase of the critical path caused by the added multiplexers/demultiplexers is compensated for completely by getting rid of the required VNs.

Pitstop is a general-purpose scheme since (1) we stress test Pitstop by increasing injection rates and the number of pipeline stages, and introducing faulty links for increased congestion to induce more deadlocks, and yet Pitstop's saturation point still matches state-of-the-art schemes, (2) it outperforms state-of-the-art schemes for PARSEC and SPLASH-2 while using no VNs, and (3) Pitstop's configuration for determining *root routers* can be tuned at boot time to better handle applications that are deadlock prone, while still guaranteeing both types of deadlock freedom. For example, if it is known that deadlocks are frequent at certain points in the network (e.g., the central routers of a mesh), then the *root* switching path can pass through the deadlock prone areas more often to increase the probability of relieving the deadlocks.

VII. Related Work

Deadlocks solutions can be classified into *avoidance*, *detection and recovery*, and *periodic recovery* approaches.

Deadlock Avoidance: Applying turn restrictions to the routing algorithm is the simplest way to avoid cyclic dependencies [15], [37]. Although simple, this technique reduces path diversity and can degrade performance and fault tolerance. Escape VCs [18]–[20] can mitigate this limitation. Turn restrictions are only placed on the Escape VC which guarantees deadlock freedom provided all packets have equal opportunity to access the Escape VC. Non-escape VCs use adaptive routing to restore path diversity. Escape VCs are a costly solution as VCs add extra area and power overhead and are often underutilized even for extreme cases [2], [13]. Ebda [22] uses fully adaptive routing across all VCs, however, turn restrictions are applied to each of them. Applying restrictions to packet injection also avoids deadlock [11], [14]. These techniques are limited to ring-based topologies such as tori; hence, they cannot support irregular topologies or faulty networks. Other approaches rely on misrouting packets [37], [48]. Misrouting can increase congestion and link utilization which translates to higher energy consumption. Some solutions use re-injection and reusing concepts like Pitstop [30], [31], [42]. In-transit buffers temporarily eject packets and then inject them using temporary host workstations to avoid network deadlock at the cost of non-minimal routing [30], [31]. Elastic buffers [42] use the existing storage in pipelined channels instead of VCs.

To avoid protocol-level deadlock, messages of different types must be placed into different networks (virtual or physical) to prevent blocking. Physical networks offer higher bandwidth while VNs [44] save on physical links but still impose significant overheads due to the additional VCs [12], [13], [39]. Bubble Coloring (BC) [64] handles both deadlocks by reserving a packet-sized bubble per message type in a virtual ring plus one bubble per router. This scheme suffers from frequent misroutes, degrading the worst-case latency (Sec. III-B) and performance [5]. In terms of performance, BC is comparable to SWAP [48] since they both misroute packets; however, SWAP has fewer misrouted packets.

Deadlock Detection and Recovery: SPIN [55] probes the network to detect possible deadlock. Once the deadlock path is confirmed, SPIN synchronizes the routers within the deadlock chain to release the frozen packets at the same time and then resumes normal execution. SPIN increases the router complexity to manage false negatives and positives, limiting its scalability. Static Bubble [56] leverages ideas from Bubble Flow Control [14], [53] to reserve one free buffer in any ring/torus topology to avoid a deadlock. These works suffer from complex control hardware due to the sending probe packets [55] and timeout counters [3], [55], [56], [60] which are needed to detect deadlock. Recovery mechanisms come at the cost of either additional buffers [3], [56], [60] or synchronizing routers to make forward progress [55]. As an example, mDisha [60] dedicates two extra buffers within the NI (in addition to injection and ejection queues) and one extra buffer within each router to handle both type of deadlocks. While mDisha can work without separate queues in the NI, it uses them for performance [60]; as a result, mDisha has more buffer resources than Pitstop. While Pitstop can resolve multiple protocol-level deadlocks, mDisha only resolves one at a time. This is because Pitstop allows one golden packet of

each message type in the network at any time while in mDisha only one deadlocked packet can be resolved. Like SPIN, mDisha also suffers from the incurred overhead caused by deadlock detection mechanisms, killing the throughput when deadlock rate increases as seen in Fig. 18 for SPIN.

Periodic Deadlock Recovery: This class of solution does not explicitly use deadlock detection [46]–[49]. For example, SWAP [48] periodically swaps one blocked packet at the cost of misrouting another. DRAIN [46] periodically drains all packets in the network to recover from potential deadlocks. Hence, DRAIN may significantly increase worst-case latency and it also needs to use multiple VNs to avoid a large amount of buffer space required for providing protocol-level deadlock freedom. Both DRAIN and SWAP need to support U-Turns increasing the complexity of the crossbar. They also need packet truncation support for wormhole flow control.

VIII. Conclusion

Most prior techniques to handle protocol and network-level deadlocks are overprovisioned especially considering how rarely deadlocks occur. The use of virtual networks, complex deadlock hardware, and misrouting leaves power, area and latency savings on the table. We propose Pitstop, a low-cost technique that reuses existing NoC resources to handle both protocol- and network-level deadlocks, without the use of virtual networks. Compared to the state-of-the-art solutions, Pitstop can improve performance up to 11% and reduce power and area up to 41% and 40%.

Acknowledgments

We thank the anonymous reviewers and members of the NEJ Group for their helpful comments to improve this work. We thank the SPIN and SWAP authors for sharing their gem5 implementations. The authors 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] F. Alazemi, A. AziziMazreah, B. Bose, and L. Chen, "Routerless network-on-chip," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[3] 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.

[4] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "Openpiton: An open source manycore research framework." Association for Computing Machinery, 2016.

[5] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler, "Slim NoC: A low-diameter on-chip network topology for high energy efficiency and scalability," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 43–55.

[6] 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.

[7] 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.

[8] C.-H. Chen, "Design and implementation of low-latency, low-power reconfigurable on-chip networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2016.

[9] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The ibm blue gene/q interconnection fabric," *IEEE Micro*, vol. 32, no. 1, pp. 32–43, 2012.

[10] L. Chen and T. M. Pinkston, "Nord: Node-router decoupling for effective power-gating of on-chip routers," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 270–281.

[11] 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.

[12] L. Chen, L. Zhao, R. Wang, and T. M. Pinkston, "Mp3: Minimizing performance penalty for power-gating of clos network-on-chip," in *IEEE Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2014.

[13] L. Chen, D. Zhu, M. Pedram, and T. M. Pinkston, "Power punch: Towards non-blocking power-gating of NoC routers," in *IEEE Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2015, pp. 378–389.

[14] L. Chen and T. M. Pinkston, "Worm-bubble flow control," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[15] 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.

[16] M. Daneshtalab, M. Ebrahimi, P. Liljeberg, J. Plosila, and H. Tenhunen, "Memory-efficient on-chip network with adaptive interfaces," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, pp. 146–159, 2012.

[17] B. K. Daya, C. O. Chen, S. Subramanian, W. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L. Peh, "Scorpio: A 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 25–36.

[18] 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.

[19] 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.

[20] 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.

[21] J. Duato, S. Yalamanchili, and N. Lionel, *Interconnection Networks: An Engineering Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[22] M. Ebrahimi and M. Daneshtalab, "EbDa: A new theory on design and verification of deadlock-free interconnection networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[23] N. Enright Jerger, T. Krishna, and L.-S. Peh, *On-Chip Networks: Second Edition*, M. Martonosi, Ed. Morgan Claypool, 2017.

[24] C. Fallin, C. Craik, and O. Mutlu, "Chipper: A low-complexity bufferless deflection router," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, 2011.

[25] H. Farrokhbakht, H. M. Kamali, and N. Enright Jerger, "Muffin: Minimally-buffered zero-delay power-gating technique in on-chip routers," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[26] H. Farrokhbakht, H. M. Kamali, N. Enright Jerger, and S. Hessabi, "Sponge: A scalable pivot-based on/off gating engine for reducing

static power in noc routers," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2018, pp. 1–8.

[27] H. Farrokhbakht, H. M. Kamali, and S. Hessabi, "SMART: a scalable mapping and routing technique for power-gating in noc routers," in *International Symposium on Networks-on-Chip (NOCS)*, 2017, pp. 1–8.

[28] H. Farrokhbakht, M. Taram, B. Khaleghi, and S. Hessabi, "Toot: an efficient and scalable power-gating method for noc routers," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2016, pp. 1–8.

[29] H. Farrokhbakht, H. Kao, and N. Enright Jerger, "UBERNoC: Unified buffer power-efficient router for network-on-chip," in *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, 2019.

[30] J. Flich, M. P. Malumbres, P. Lopez, and J. Duato, "Improving routing performance in Myrinet networks," in *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, 2000.

[31] J. Flich, P. López, M. P. Malumbres, J. Duato, and T. Rokicki, "Combining in-transit buffers with optimized routing schemes to boost the performance of networks with source routing," in *Proceedings of the Third International Symposium on High Performance Computing*, 2000.

[32] E. K. Gawish, M. W. El-Kharashi, and M. Abu-Elyazeed, "Process variability-induced NoC link failure," *Microelectron. J.*, vol. 46, no. 3, pp. 248–257, 2015.

[33] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011, pp. 350–361.

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

[35] S. M. Hassan and S. Yalamanchili, "Centralized buffer router: A low latency, low power router for high radix nocs," in *2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, 2013.

[36] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, 2011.

[37] S. A. R. Jafri, Y.-J. Hong, M. Thottethodi, and T. N. Vijaykumar, "Adaptive flow control for robust performance and energy," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43, 2010.

[38] G. Kim, H. Choi, and J. Kim, "TCEP: Traffic consolidation for energy-proportional high-radix networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 712–725.

[39] J. Kim, "Low-cost router microarchitecture for on-chip networks," in *42nd IEEE/ACM International Symposium on Microarchitecture*, 2009.

[40] H. Kwon and T. Krishna, "Opensmart: Single-cycle multi-hop noc generator in bsv and chisel," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 195–204.

[41] D. Lee, R. Parikh, and V. Bertacco, "Brisk and limited-impact NoC routing reconfiguration," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014.

[42] G. Michelogiannakis, J. Balfour, and W. J. Dally, "Elastic-buffer flow control for on-chip networks," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, 2009, pp. 151–162.

[43] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009.

[44] 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.

[45] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "Vichar: A dynamic virtual channel regulator for network-on-chip routers," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] R. Parikh and V. Bertacco, "Formally enhanced runtime verification to ensure noc functional correctness," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 410–419.

[51] R. Parikh and V. Bertacco, "uDIREC: Unified diagnosis and reconfiguration for frugal bypass of NoC faults," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013.

[52] R. Parikh, R. Das, and V. Bertacco, "Power-aware NoCs through routing and topology reconfiguration," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014.

[53] 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.

[54] R. S. Ramanujam, V. Soteriou, B. Lin, and L. Peh, "Design of a high-throughput distributed shared-buffer noc router," in *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, 2010.

[55] 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.

[56] 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.

[57] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar, "A 2 tb/s $6 \times$ 4 mesh network for a single-chip cloud computer with dvfs in 45 nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 4, pp. 757–766, 2011.

[58] K. Sankaralingam, R. Nagarajan, R. McDonald, R. Desikan, S. Drolia, M. S. Govindan, P. Gratz, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, S. W. Keckler, and D. Burger, "Distributed microarchitectural protocols in the trips prototype processor," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006, pp. 480–491.

[59] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker, "Autonet: a high-speed, self-configuring local area network using point-to-point links," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1318–1335, 1991.

[60] 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.

[61] D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, 1st ed. Morgan Claypool, 2011.

[62] A. T. Tran and B. M. Baas, "Roshaq: High-performance on-chip router with shared queues," in *IEEE 29th International Conference on Computer Design*, 2011, pp. 232–238.

[63] A. T. Tran, D. N. Truong, and B. Baas, "A reconfigurable source-synchronous on-chip network for gals many-core platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 897–910, 2010.

[64] 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.

[65] 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.

[66] J. Yin, O. Kayiran, M. Poremba, G. Loh, and N. Enright Jerger, "Efficient synthetic traffic models for large, complex SoCs," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2016.